A Queueing Analysis Of RAID Architectures

Shenze Chen and Don Towsley
Department of Computer and Information Science
University of Massachusetts
Amherst, MA 01003

COINS Technical Report 91-71
September, 1991

A Queueing Analysis of RAID Architectures *

Shenze Chen *Don Towsley*

Department of Computer and Information 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. Removing this bottleneck has proven to be a challenging research problem. One proposed solution is the so called "disk array", consisting of multiple disks with data spread over these disks. In this paper, we examine the performance of two of the most promising RAID (Redundant Arrays of Inexpensive Disks) architectures, the *mirrored array* and the *rotated parity array*. Mathematical models are presented for the analysis of the two architectures. These architectures are compared under two kinds of workloads: (a) small I/O as found in transaction processing or UNIX environment; and (b) large I/O as found in scientific computations or image processing. The main difference between this study and that of previous work is that we propose and evaluate the performance of a more efficient scheduling policy for the mirrored array architecture. As a consequence, the analytic results show that, given the same number of disks, the mirrored array coupled with the new policy 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. This conclusion differs from that of previous studies, due to the efficiency of the proposed policy for the mirrored array. Finally, simulation results are presented which substantiate our conclusions.

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-796, 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, 8], whereas disk seek times have only improved by 7% per year and the rotation times not at all [7, 23]. This has led to a big gap between the speed of the CPU/main memory and that of the disk I/O subsystem. This gap is expected to increase in the near future. Currently, disks are at least four orders of magnitude slower than main memory [19]. Based on these observations, we predict that further increases in the processor speed will bring little gain to the overall system performance. Therefore, a fundamental problem is: based on the current foreseeable future disk technology, how to 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 [10] and Salem et al [28], respectively. Since then a great deal of work has focused on various design issues related to the performance of disk arrays [19, 20, 11, 13, 26, 3, 32] and to their reliability [30, 6, 16]. Disk array architectures fall into one of five different classes proposed in [24, 25, 9] 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 a new scheduling policy, called *Shortest Queue Policy with Minimum Seek (SQP-MS)* for the mirrored disk array which performs considerably better than the simple *minimum seek policy (MSP)* that was used in conjunction with RAID 1 in previous studies [22, 5]. Unlike MSP, SQP-MS allows different requests to simultaneously access a data item thus providing a significant bandwidth improvement and reduction in response times. In order to distinguish between the mirrored array architecture operating under the two policies, we will refer to the mirrored array combined with SQP-MS as RAID 1+. Mathematical models for estimating the response times of the mirrored disk array coupled with MSP, the traditional algorithm, and our first proposed algorithm SQP-MS, and RAID 5 are constructed. We also examine the maximum throughput of RAID 1+ and RAID 5. Two kinds of workloads are used to compare these architectures: (1) small I/O which is typical in many transaction processing applications and UNIX environments; and (2) large I/O which can be found in scientific or supercomputing applications. The results show that in most cases, even for applications where the I/O request size is large, RAID 1+ (with SQP-MS) outperforms RAID 5, given the same number of disks for both architectures. This conclusion is quite different from that of [5] where RAID 5 was found to outperform RAID 1 (with MSP) for applications with predominantly large I/O requests. Obviously, a main drawback of mirrored arrays is that the useful capacity is reduced by 50%. Therefore, two options are available to users when given N disks, namely, they can opt for either a higher performance but less capacity by configuring the N disks to a mirrored array, or a lower performance but larger capacity via RAID 5.

Other related work on performance evaluation of disk arrays can be found in [14, 27, 21]. However, disk arrays examined in these works do not include redundant information to provide reliable I/O subsystems.

I/O performance can also be improved by introducing a disk cache [29, 17]. 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 I/O subsystem model. The analyses of the mirrored disk arrays are given in Section 3. Section 4 includes the analysis of the RAID 5 architecture. The performance of these two architectures is compared with each other in Section 5. Last, section 6 summarizes this paper.

2 RAID Architectures and the I/O Subsystem Model

In this section, we first describe the two RAID architectures of interest to us. Then we describe the I/O subsystem model used in our discussions.

2.1 RAID Architectures

We consider disk arrays of N identical disks. In the architectures of interest to us, data is *block interleaved* across W disks, where W is the *strip 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 strip (n might be larger than the strip width W). As will be observed later the strip width is $N/2$ for our mirrored array architecture, and $N-1$ for the RAID 5 architecture. 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* [10, 9]. 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. By interleaving data across strips in an array, we can effectively distribute the burden of heavily used files among multiple disks.

2.1.1 The Mirrored Array Architecture

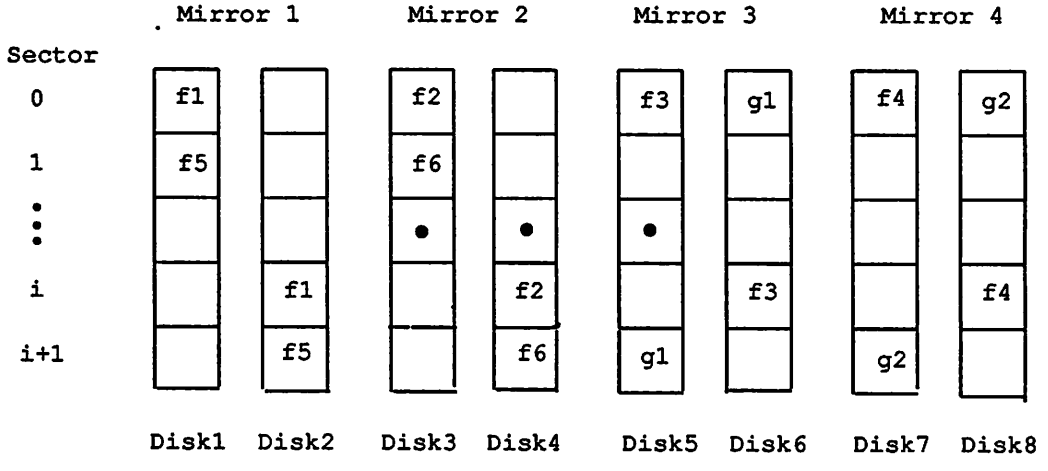


Figure 1: The RAID 1 Architecture ($N = 8$).

In the mirrored disk array, 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 stripped 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 $N/2$–stripped disk pairs. Figure 1 shows a scenario where two files, f consisting of 6 blocks and g consisting of 2

blocks, are allocated on an array of 8 disks. A read request can be satisfied by either copy, but a write must be performed on both copies.

Scheduling Policies for the Mirrored Array Architecture :

Since there are two copies of data stored in a mirrored array and reads can be satisfied by either copy, there are different ways to schedule read requests to one of these two copies.

- **The MSP Policy:**
  The *minimum seek policy (MSP)* used in other studies [5, 22] behaves in the following way: each arrival request joins a single queue which is served in a FIFO order. If the request in front of the queue is a read, it is served by the disk where the head is closest to the cylinder containing the data. If it is a write, it starts on both disks.

- **The SQP-MS Policy:**
  We propose a new policy, called the *shortest queue policy with minimum seek (SQP-MS)*, which works as follows: it maintains two separate queues, one for each data copy. Both queues are served in a first come first serve (FCFS) manner. When a write request arrives, it generates two tasks, one for each queue, and it completes when both of these two tasks finish. A read request simply joins the shortest queue at the moment of its arrival and remains there until it is served. If a read request, say R^*, arrives and finds both disks idle, it is assigned to the one that requires the least arm movement. We further identify two variations of the SQP-MS scheduler, which differ slightly in their treatment of those read requests that arrive prior to the completion of R^* and find the two queue lengths to be equal. The first variation, termed *intelligent scheduler (IS)*, requires that the scheduler maintain a one bit flag to keep track of the disk currently serving R^*. Whenever the two queue lengths are the same, a new read is always assigned to the queue associated with the server currently occupied by R^*. Since R^* arrived earlier and was assigned to the disk providing the smallest seek time, it is expected to complete before the request being served on the other disk.

  The second variation, called *simple scheduler (SS)*, does not keep track of which disk is currently serving R^*. Instead, a new read is randomly assigned to either of the two queues with equal probability whenever their lengths are the same. Since the analytic results show that the performance differences between these two variations are very small, the *simple scheduler* is preferred.

Notice that by using the SQP-MS, RAID 1+ can serve two read requests simultaneously, whereas for RAID 1 with the MSP, only one request is served at any point in time.

2.1.2 The RAID 5 Architecture

The RAID 5 architecture [9, 24, 25], also called the *rotated parity array*, is configured as in Figure 2 for the case of 8 disks.

As shown in the figure, each strip includes a redundant parity block, which is obtained by *XOR*-ing all blocks from other disks in the same strip. Parity blocks are rotated among all disks in the array. The advantages of using the rotated parity block have been discussed extensively in the literature [9, 24, 25]. When any disk fails, its data can be reconstructed from the other disks. The I/O subsystem becomes unavailable only when a second disk fails before the first failed disk has recovered.

By using this fault tolerant technique, read operations can be performed as usual. However, write operations need to update, in addition to the data blocks to be written, the corresponding

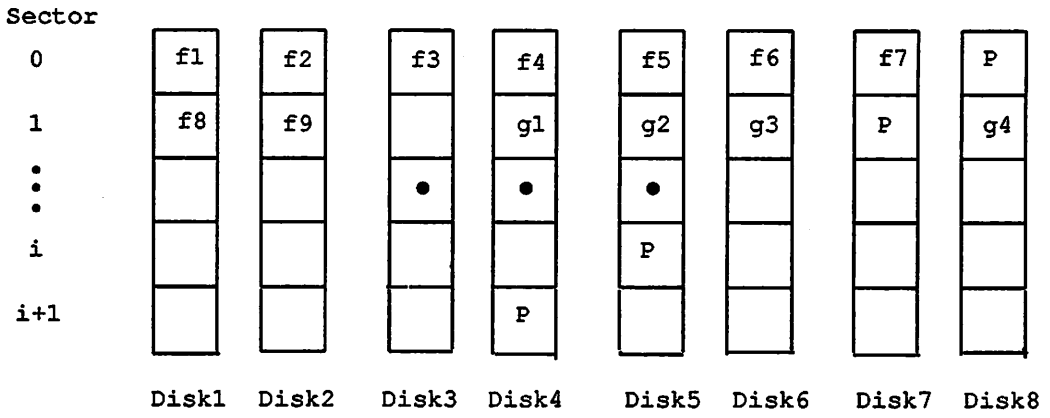


Figure 2: The RAID 5 Architecture ($N = 8$).

parity blocks. If a write request is to update all blocks in a strip, the new parity block can be calculated by *XOR*-ing all new data blocks of that strip. However, if a write is to a partial strip, the calculation of the new parity block needs to use information from the existing data blocks and the parity block from the same strip. Therefore, a *partial strip write* must first perform a read and then an update operation.

Suppose a write is to access N_s strips plus a partial strip tail of T_s blocks ($0 < T_s < W$) as shown in Figure 3. We consider two strategies to calculate the new parity block for the partial strip.

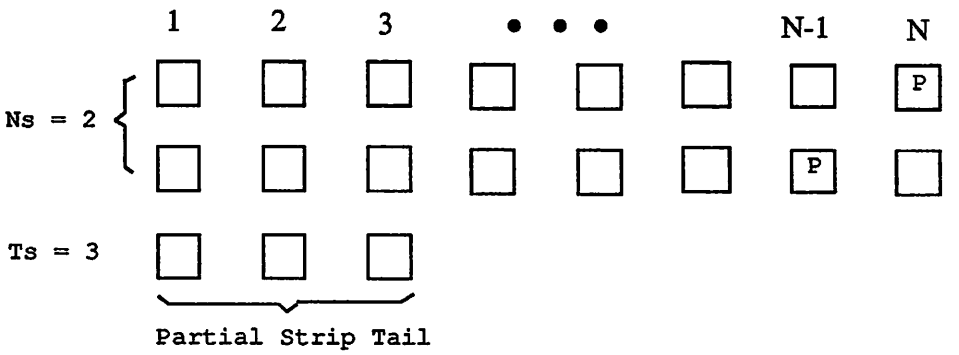


Figure 3: A Write Request with a Partial Strip Tail.

- **The RT Strategy:**
  The first strategy, called *RT*, reads all the old tail blocks and the parity block, *XOR*s them with the new tail blocks to be written, and then writes them back to the disks. Hence, the T_s disks containing the tail blocks and the disk containing the parity block pass through service stages illustrated in Figure 4.

  Here we assume that after the transfer of N_s blocks to the disk (first three stages), the time required for the controller to set up a read operation is negligible so that the transfer of a

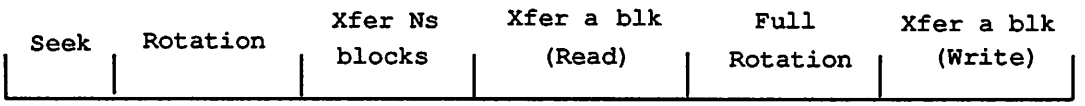
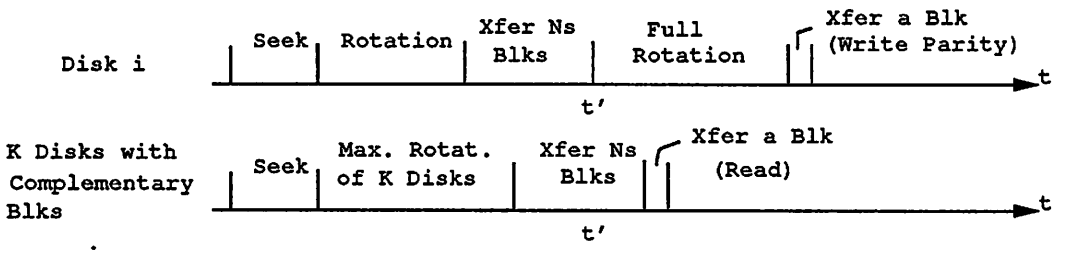


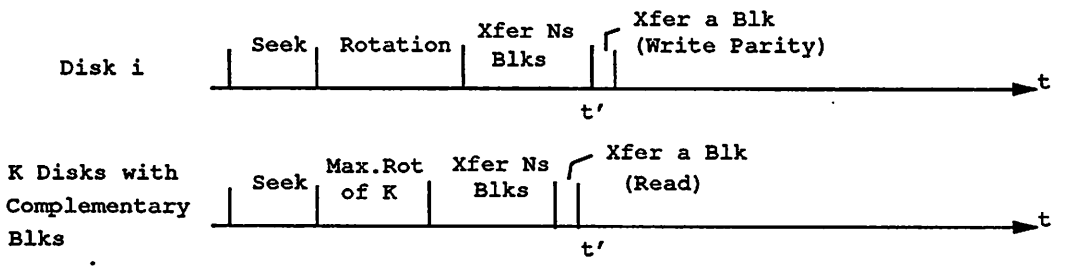
Figure 4: Service Stages of Partial Disks.

block from disk to main memory (or disk cache) can proceed immediately. This assumption leads to an optimistic estimate of the performance of RAID 5.

- **The RC Strategy:**
The second strategy, RC, reads the complementary blocks of the tail, i.e., the $W - T_s$ blocks in the strip not to be written, and then XORs them with the new tail blocks to get the new parity block. By using this strategy, all disks but the one containing the parity block of the partial strip, say disk i, will experience the first four stages in Figure 4 (the fourth stage for the T_s disks containing the tail blocks is a write instead of a read). Let t' denote the time that the disk i finishes the first three stages: seek, rotation, and transfer of N_s blocks. If disk i, at time t', finds that the new parity block has not been calculated, it performs a full rotation followed by a write of the parity block (Figure 5 (a)). Otherwise, if all of the



(a). Disk i needs a full rotation before writing the new parity block



(b). No additional full rotation needed

Figure 5: Scenarios of Serving a Write Request with a Partial Strip Tail under RC.

disks containing the complementary blocks of the tail have completed the first three stages and have also read out the block needed for calculating the new parity block by time t', then

disk i needs no additional full rotation and immediately writes the new parity block (Figure 5 (b)).

Observe that although the probability of omitting the full rotation on disk i above is usually small, strategy *RC* may still provide better performance than *RT*. This is because under RT, T_s+1 disks are required to perform an additional full rotation whereas under RC, only disk i (the disk containing the parity block of the partial strip) may be required to perform a full rotation. The RC strategy is beneficial for large I/O environments where most requests access at least one strip of data and I/O requests are served sequentially, since reading the complementary blocks of the tail will not degrade the response time even for small tails. However, this strategy is certainly not good for small I/O environments, since it requires all disks in a strip to serve each write request and therefore dramatically reduces the number of requests which can be executed concurrently. In the following discussions, for small I/O applications we always choose strategy *RT*, and for large I/O applications we will examine both the *RT* and the *RC* strategies.

Another strategy would dynamically choose to read either the tail or its complement depending on which contains less blocks. Based on the previous discussion, we expect its performance to fall between the *RC* and the *RT* strategies for large I/O applications.

2.2 I/O Workloads and Mean I/O Service Times

In order to evaluate the performance of different disk array architectures, we choose two kinds of workloads representing two extreme application areas. On one hand, in the *small* I/O case, we assume that each I/O request accesses 4096 bytes of data, which corresponds to the size of a disk block. This kind of small I/O access pattern is typically found in many transaction processing systems and UNIX environments. On the other hand, in the *large* I/O case, we assume the request size to be a *r.v.* with mean 1M bytes (256 blocks when the block size is 4096 bytes). This scenario can be found in supercomputing or image processing application areas. Other applications are expected to fall in between these two extremes.

The remainder of this section contains a derivation of the mean disk service time, $1/\mu$, which will be used in our later analyses. We assume that each disk in the array has its own data path so that the channel is not a bottleneck, and therefore the channel delay is ignored. Since it has been observed that the disk arm doesn't move [15, 11] much of the time, we introduce p_s, the *sequential access probability*, 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.

Define,

C: the number of cylinders for each disk;

D: a *r.v.* denoting the disk seek distance for each request;

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.* denoting the disk seek time;

R: a *r.v.* denoting the rotational latency to serve a request;

T: a *r.v.* for the data transfer time.

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

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

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

Removal of the conditioning yields

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

We use the following model for seek time [2, 31],

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

where a is the arm acceleration time (3 ms), and b is the seek factor (0.5) [21]. The mean seek time is

$$\begin{aligned} E[S] &= \sum_{i=1}^{C-1} P\{D=i\}(a+b\sqrt{i}) \\ &= (1-p_s)\left[a+b\frac{2}{C-1}\left(\sum_{i=1}^{C-1} i^{\frac{1}{2}} - \frac{1}{C}\sum_{i=1}^{C-1} i^{\frac{3}{2}}\right)\right]. \end{aligned}$$

By approximating the summations by integrals, e.g., $\sum_{i=1}^{C-1} i^{\frac{1}{2}} = \int_1^{C-1} x^{\frac{1}{2}}\,dx$, we obtain

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

The rotational latency for a single disk is assumed to be uniformly distributed in the interval $[0, 16.7]$, $P\{R \le x\} = \frac{x}{16.7}$. Hence, $E[R] = 8.3$.

Finally, the mean disk service time is

$$\frac{1}{\mu} = E[S] + E[R] + E[T]. \tag{2}$$

We will observe later on that the statistics for S will be insensitive to the size of the I/O request and the type of disk array. However, the statistics of R and T vary from architecture to architecture and depend on the I/O request size. The variations from a single disk will be treated in later sections.

Other disk parameters used in our model include: number of cylinders $C = 1200$; maximum transfer rate $= 3Mb/sec$. Hence the transfer time for a single block (4096 bytes) is 1.3 ms. In [11], Kim reported several real disk reference traces, which show the percentage of sequential accesses ranging from 25% to 50%. Here we assume the sequential access probability p_s to be 0.5 in our analysis. We will also vary p_s to investigate its performance impact. With these parameter settings, the average seek time is 6.1 milliseconds.

3 A Queueing Analysis of Mirrored Disk Arrays

In an earlier paper [33], we evaluated the performance of MSP and SQP (*shortest queue policy*, which differs from the SQP-MS policy in that a read randomly chooses either disk when both are idle) via simulations, and showed that SQP outperforms MSP for a system consisting of two mirrored disks.

In this section, we present approximate analytic models for the proposed RAID 1+ architecture which uses the SQP-MS policy for both small I/O and large I/O applications. We will also briefly discuss the analysis of the traditional mirrored array which employs MSP to provide a baseline for comparisons.

As stated above, the I/O subsystem model consists of an array of N disks. Arrivals to the I/O subsystem are generated by a Poisson process with rate λ. A request is a read with probability p_r, and a write with probability $p_w = 1 - p_r$. In order to make the analysis tractable, the disk service time is assumed to be exponentially distributed with mean $1/\mu$, as calculated in the last section.

We introduce the following notation,

N: the number of disks in the array;

α: the transfer time for a single block;

W: the strip width;

p_r, p_w: the probabilities that a request is a read or write, $p_r + p_w = 1$;

p_f: the probability that a large I/O performs a full strip operation.

R_j: a *r.v.* for the rotational latency of disk j;

X: the sum of rotational latency and transfer time to serve a request;

N_s: a *r.v.* for the number of strips in a large I/O request;

T_s: a *r.v.* for the number of tail blocks in a large request;

L: a *r.v.* for request size (blocks) for large I/O applications, $L = N_s W + T_s$;

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.

3.1 RAID 1+ for Small I/O Applications

In transaction processing applications or UNIX environments, the I/O access pattern is characterized by multiple I/O requests, each accessing a small amount of data. Hence, we assume the I/O request size to be 1 block (4096 bytes), which is equal to the unit of interleaving for our disk arrays. In this case, a read request needs only to access one disk, whereas a write request has to update a pair of disks. Consequently, the N-disk array configured as RAID 1+ under policy SQP-MS can support up to N read or $N/2$ write requests simultaneously.

The RAID 1+ subsystem model for the small I/O applications is illustrated in Figure 6. We assume that I/O requests arrive at the subsystem according to a Poisson process with rate λ. The I/O subsystem dispatcher routes each request to the disk pair where the target block is stored. Let p_i be the probability that a request is routed to *mirror i*. We assume that a request will be directed to one of the $N/2$ mirrored pairs with equal probability, i.e., $p_i = 2/N, i = 1, 2, ..., N/2$. Consequently,

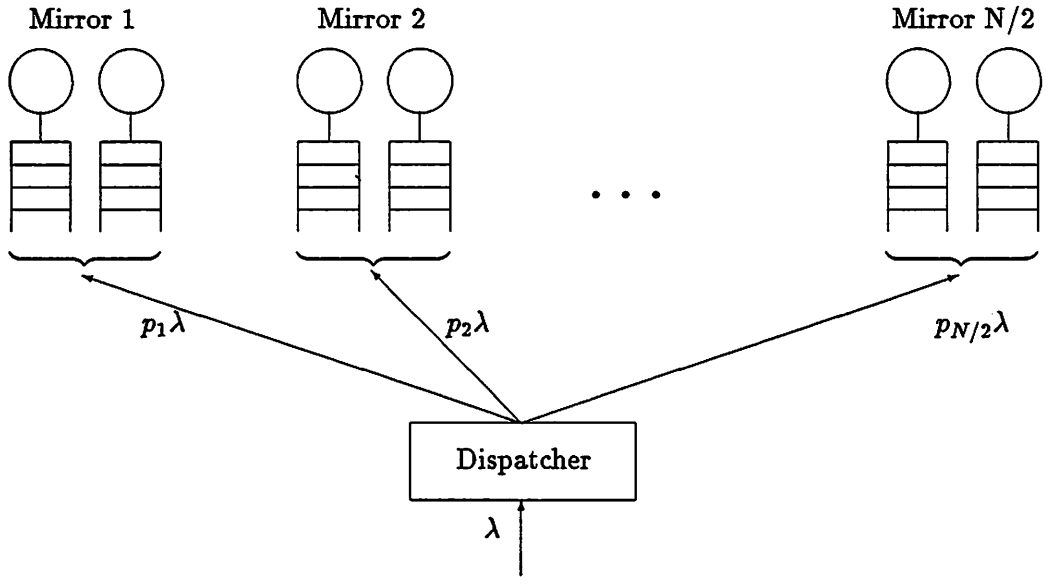


Figure 6: RAID 1+ Queueing Model for Small I/O Applications

all of the $N/2$ mirrored pairs are statistically identical with requests arriving according to a Poisson process with parameter $2\lambda/N$. Thus we only need to focus on one pair of mirrored disk in our analysis.

The behavior of the mirrored disk under the *intelligent SQP-MS scheduler* (see Section 2.1.1) is modeled by a three dimensional Markov chain, $\mathbf{N}(t) = (N_{max}(t), N_{min}(t), N_3(t))$, where the state variables are defined as,

$N_{max}(t)$: the maximum queue length at time $t > 0$,

$N_{min}(t)$: the minimum queue length at time t,

$N_3(t)$: to distinguish the situation where the most recent busy period prior to time t was initiated by a read and this read is still in service ($N_3(t) = 1$) from other cases, i.e., either the most recent busy period was initiated by a write or by a read that has already completed ($N_3(t) = 0$).

The *simple scheduler* (Section 2.1.1) model is treated in a similar way but requires $N_3(t)$ to take an additional value and a slightly different meaning as described in Appendix A. Since both $N_{max}(t)$ and $N_{min}(t)$ are unbounded $r.v.$'s, it is intractable to solve the Markov chain exactly. Instead, we truncate one of these $r.v.$'s, say $N_{min}(t)$, so that it never exceeds a finite value B in order to obtain lower bounds for the performance measures of interest. Whenever the shortest queue is full and a read request arrives, it passes through without delay and whenever a write request arrives, it generates one task that joins the longest queue and a second that passes through the shortest queue without delay. The write completes as soon as that task finishes service. On the other hand, in order to achieve an upper bound, we slightly modify the definition of the Markov chain to $\mathbf{N}(t) = (N_{min}(t), \Delta(t), N_3(t))$, where $\Delta(t) = N_{max}(t) - N_{min}(t)$ is truncated to B. Now whenever $\Delta(t) = B$ and a task finishes service on the disk with the shortest queue, a fictitious task is generated and occupies that disk, so that the system state is unchanged.

This method for generating bounds is similar to that used to establish bounds for fork/join queueing models introduced in an earlier paper [34] and is omitted here. In fact, when we choose

$B = 32$, the bounds are observed to be very tight so that the difference occurs only when the device utilization exceeds 90%. The stationary performance metrics of these two systems can be obtained by using the matrix-geometric method [18]. The detailed calculations for solving the lower bound system is given in Appendix A for the two variations of the SQP-MS scheduler. Calculations for the upper bound system are similar and therefore omitted. The mean response times for both read and write requests are obtained by taking the average of the lower and upper bounds. The disk service time is assumed to be exponentially distributed with mean $1/\mu$, where $1/\mu$ is obtained from equations (1) and (2). Given the disk parameters specified in Section 2, the mean rotational latency $E[R] = 8.3$ ms, and mean transfer time $E[T] = 1.3$ ms for small I/O applications.

Read requests that find both data copies available when they arrive at the I/O subsystem, are assumed to have exponentially distributed disk service times with mean $1/\mu'$, which is obtained as follows: for such a read, the seek distance is a *r.v.* defined by $D_{min} = \min\{D^{(1)}, D^{(2)}\}$, where $D^{(1)} =_d D^{(2)} =_d D$. (Here "$=_d$" denotes "equal in distribution".) The distribution of D_{min} is

$$P\{D_{min} = i\} = P\{D^{(1)} = i, D^{(2)} > i\} + P\{D^{(1)} > i, D^{(2)} = i\} + P\{D^{(1)} = i, D^{(2)} = i\}.$$

Since $D^{(1)}$ and $D^{(2)}$ are independent of each other, we have

$$P\{D_{min} = i\} = 4(1 - p_s)^2 \frac{(C - i)^3}{C^2(C - 1)^2}, \qquad i > 0$$

and

$$P\{D_{min} = 0\} = p_s(2 - p_s).$$

In a similar way as above, define $S_{min} = \min\{S^{(1)}, S^{(2)}\}$ which has mean

$$\begin{aligned} E[S_{min}] &= \sum_{i=1}^{C-1} P\{D_{min} = i\}(a + b\sqrt{i}) \\ &\approx (1 - p_s)^2 \left[a + b\sqrt{C - 1} \times 0.102\right] \end{aligned} \tag{3}$$

and the mean service time, $1/\mu'$, for these read requests are immediately obtained from equation

$$\frac{1}{\mu'} = E[S_{min}] + E[R] + E[T]. \tag{4}$$

3.2 RAID 1+ for Large I/O Applications

Supercomputing or image processing systems usually carry lower multiprogramming levels and, as a consequence, 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 in-memory data structures, and results are periodically written back to disks [9]. To model these scenarios, we assume the I/O request size to be a *r.v.* L (in blocks) having an arbitrary distribution. We further assume that each I/O request accesses at least one strip of data. Let W be the strip width, which is $N/2$ for RAID 1+. A request of L blocks can be uniquely expressed as $L = N_s W + T_s$, where N_s is the number of strips and T_s is the tail, $T_s < W$. Because of the I/O size, all disks in a strip are involved in serving each I/O request. For example, if there are a total of 64 disks in the array and the interleaved block size is 4096 bytes, then the strip width for the RAID 1+ architecture is 32, and an I/O request of 200 blocks will access 6 strips plus a partial strip of 8 blocks. Thus 8 disks are accessed for 7 blocks each and the remaining disks provide 6 blocks each.

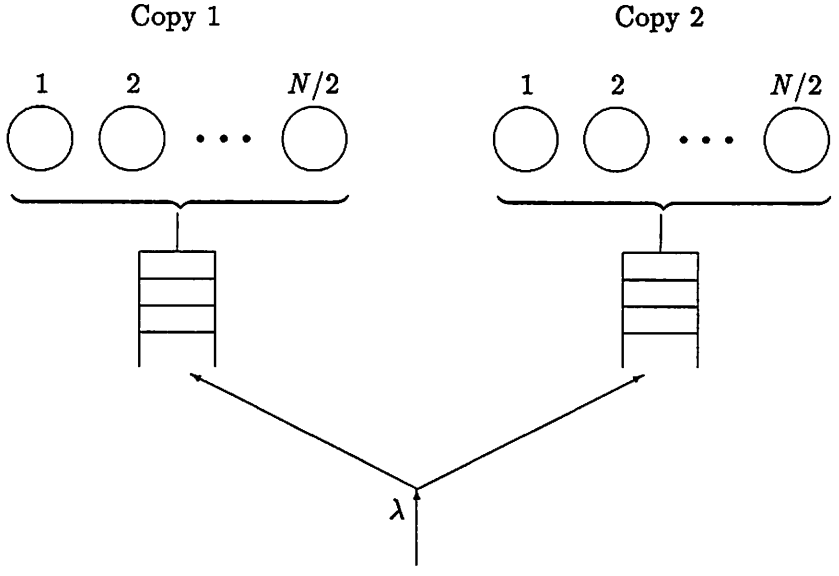


Figure 7: RAID 1+ Queueing Model for Large I/O Applications

The queueing model is shown in Figure 7. In this model, the N disks are divided into two groups, each with $N/2$ disks. Each group keeps a copy of the data. The RAID 1+ subsystem maintains two queues, one for each data copy. When an I/O request in front of the queue is scheduled for service, all of the $N/2$ disks in the group are initiated simultaneously. Because of the data layout as described earlier (Figure 1), arms on all these $N/2$ disks are moved to the same cylinder. Hence the seek operations on all of the $N/2$ disks can be considered to be synchronized. Occasionally there may be one track of difference, but the impact of this difference is negligible.

The rotations of disks, however, are not synchronized [5, 11]. Thus, if we assume that the rotational latencies on the disks, R_j, $j = 1, 2, \cdots, W$, are uniformly distributed in $[0, 16.7]$ and independent of each other, and X is the time to complete the rotation and transfer on all W (which is $N/2$ for RAID 1+) disks, then,

$$X = \max\{\max_{j=1,T_s}\{R_j + \alpha\}, \max_{j=T_s+1,W}\{R_j\}\} + N_s\alpha$$

where α is the transfer time of a single block.

In the following, we obtain an expression for the expected value of X conditioned on the request size being $L = k$, $E[X|L = k]$. Write $k = mW + n$, and let

$$\tilde{X} = \max\{\max_{j=1,T_s}\{R_j + \alpha\}, \max_{j=T_s+1,W}\{R_j\}\}.$$

Observe that when $n = 0$, $\tilde{X}$ is the maximum rotational latency for W disks. When $0 < n < W$, due to the independence of rotational latency of each disk, and the assumption that R_j is uniformly distributed in $[0, 16.7]$, we have

$$P\{\tilde{X} \le x|L = k\} = \begin{cases} \left(\frac{x}{16.7}\right)^{W-n}\left(\frac{x-\alpha}{16.7}\right)^{n} & \alpha \le x \le 16.7, \\ \left(\frac{x-\alpha}{16.7}\right)^{n} & 16.7 < x \le 16.7 + \alpha. \end{cases}$$

Hence,

$$E[X|L = k] = \begin{cases} 16.7\frac{W}{W+1} + m\alpha & n = 0, \\ \int_{\alpha}^{16.7+\alpha} x\, d_x P\{\tilde{X} \le x|L = k\} + m\alpha & n = 1, 2, \cdots, W-1. \end{cases} \quad (5)$$

When the distribution of L is given, removal of the conditioning yields

$$E[X] = \sum_{k=W}^{\infty} E[X|L=k]P\{L=k\}. \tag{6}$$

Finally, the mean disk service time is

$$\frac{1}{\mu} = E[S] + E[X]. \tag{7}$$

Again, read requests which arrive and find that both copies are available can be scheduled to the group that provides the minimum seek time and, for these reads, the mean service time is

$$\frac{1}{\mu'} = E[S_{min}] + E[X]. \tag{8}$$

Based on these discussions, we conclude that the analysis techniques used for the RAID 1+ architecture for small I/O applications are also applicable here when the I/O size is large. In applying them, we only need to account for the larger mean latency and transfer time within the disk service time.

3.3 RAID 1 with Minimum Seek Policy (MSP)

In this subsection, we briefly discuss the RAID 1 architecture which uses the *minimum seek policy (MSP)* as described in Section 2.2. Since all requests are served sequentially, it can be modeled as an $M/G/1$ queue. All reads are satisfied by the copy with the minimum seek distance, and therefore the read response time is assumed to be an exponential *r.v.* with mean $1/\mu'$, where μ' is obtained from equations (3), (4) and (8). The write response time is the maximum of two exponential *r.v.*'s with mean $1/\mu$, where μ is obtained from equations (1), (2) and (7). Consequently, the overall service time density function is

$$f_Y(x) = p_r\mu' e^{-\mu' x} + p_w 2\mu e^{-\mu x}(1 - e^{-\mu x})$$

and the first and second moments are

$$\overline{Y} = p_r\frac{1}{\mu'} + p_w\frac{3}{2\mu}$$

$$\overline{Y^2} = p_r\frac{2}{\mu'^2} + p_w\frac{7}{2\mu^2}.$$

The mean rotational latency and transfer time are the same as that of in Section 3.1 and 3.2 for small I/O and large I/O applications respectively.

4 An Analysis of the RAID 5 Architecture

In this section, we present analytic queueing models for the RAID 5 architecture for both small I/O and large I/O applications.

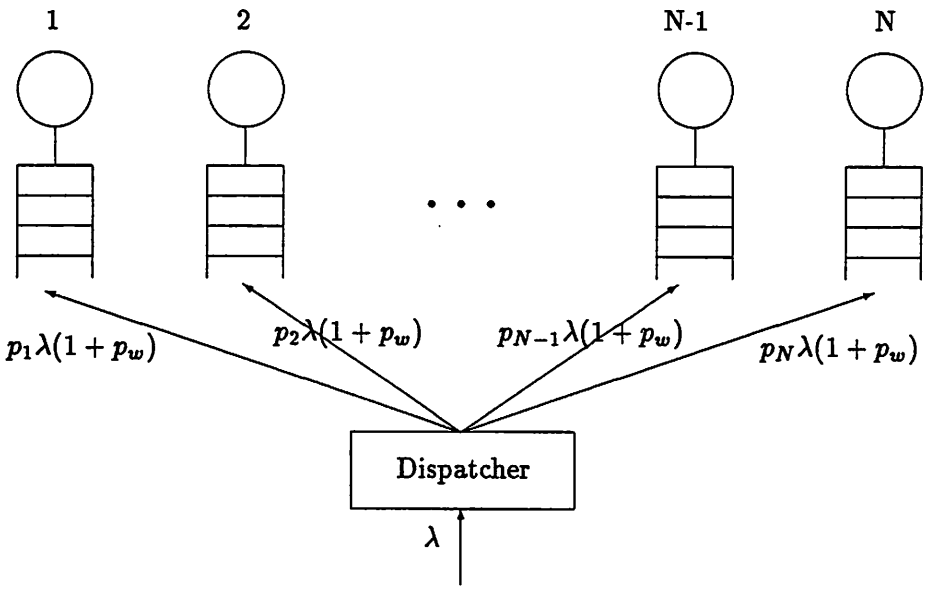


Figure 8: RAID 5 Queueing Model for Small I/O Applications

4.1 RAID 5 for Small I/O Applications

As stated before, for small I/O applications, we assume that each I/O request accesses 1 block of data. As shown in Figure 8, I/O requests arrive to the disk subsystem according to a Poisson process with rate λ. After its arrival, a request is directed to the disk containing its target data block by the I/O subsystem dispatcher. As before, each disk in the array is assumed to be accessed with equal probability, i.e., $p_i = 1/N, i = 1, 2, ..., N$. If the request is a write, then it is also directed to the disk containing the appropriate parity block in order to update the parity block.

We approximate the behavior of each disk as an $M/G/1$ queue and assume that a write request completes when the block has been written. This ignores the effects of the synchronization required to update the parity block, namely that the new parity block can not be written prior to the old value of the data block being read out. Therefore, our analytic model provides an optimistic result for the RAID 5 architecture in a small I/O environment when $p_w > 0$.

The service time for a read request, Y_r, as before, consists of seek, latency, and transfer time. Since the I/O size is assumed to be 1 block, with a $3Mb/sec$ transfer rate, the transfer time is 1.3 ms. The mean rotational latency is 8.3 ms. Hence the mean read response time, $1/\mu$, can be obtained directly from equations (1) and (2). In order to be compatible with the analysis of RAID 1+, we again assume this service time to be exponentially distributed with rate μ.

The service time for a write request consists of a seek, rotational latency, block read followed by a full rotation, and the time required to write the block back to the disk, as shown in Figure 4 with $N_s = 0$. Consequently, the service time for a write request can be expressed as,

$$Y_w = Y_r + \Delta$$

where Δ is a constant consisting of the full rotation time (16.7 ms) and the second transfer time (α).

Because of the special *read-update* procedure required for write operations in the RAID 5 architecture which adopts the *RT* strategy to calculate new parity blocks, each write request will also bring extra workload to the disk where the corresponding parity block is allocated. Let p_r be the

probability that a request is a read and p_w be the probability that a request is a write. Then, each disk will face an arrival rate of $\lambda_i = p_i\lambda(1+p_w)$. These requests to update parity blocks behave in the same way as writes, i.e., they need to follow the *read-update* procedure. Therefore, the actual fractions of reads and writes are $\tilde{p}_r = p_r/(1+p_w)$ and $\tilde{p}_w = 2p_w/(1+p_w)$ respectively.

Let Y be the *r.v.* for disk service time; it has the following density function,

$$f_Y(x) = \begin{cases} \tilde{p}_r\mu e^{-\mu x} & x < \Delta, \\ \tilde{p}_r\mu e^{-\mu x} + \tilde{p}_w e^{\mu\Delta}\mu e^{-\mu x} & x \geq \Delta \end{cases}$$

and first and second moments,

$$\overline{Y} = \tilde{p}_r\frac{1}{\mu} + \tilde{p}_w(\frac{1}{\mu} + \Delta)$$

$$\overline{Y^2} = \frac{2}{\mu^2} + \tilde{p}_w(\frac{2T}{\mu} + \Delta^2).$$

By the Pollaczek-Khinchin formula [12], the mean waiting time in the queue is

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

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

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

$$\begin{aligned} \overline{Z_r} &= \overline{Q} + \frac{1}{\mu} \\ \overline{Z_w} &= \overline{Q} + \frac{1}{\mu} + \Delta \\ \overline{Z} &= \tilde{p}_r\overline{Z_r} + \tilde{p}_w\overline{Z_w} \end{aligned}$$

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

4.2 RAID 5 for Large I/O Applications

When RAID 5 is used in a large I/O environment, all disks work together as a single logical device with a fast transfer rate (Figure 9). This can also be modeled as an $M/G/1$ queue. For read requests, the seek, latency, and transfer time can be calculated in the same way as described in the case of RAID 1+ for large I/O applications, except that the strip width W here increases from $N/2$ to $N-1$ and therefore the transfer time decreases almost by half.

For write requests, since a large I/O operation may involve several strips of data, we distinguish between two cases. In the case of a *full strip write*, where the data blocks to be updated start and end at strip boundaries, the write service time is the same as that of a read because the parity block of each strip 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 strip boundary, a *partial strip write* occurs and a $read-update$ procedure has to be followed. For simplicity, we assume a partial strip write is only required for the last strip of each I/O, i.e., the starting block is always aligned with the strip boundary [5].

In the following discussion, we use the term "partial strip write" to refer to a write request which contains N_s strips plus a partial strip tail of T_s blocks, where $T_s > 0$. Since we assume the request size L to be a *r.v.* with an arbitrary distribution, a full strip I/O occurs only when L is divisible by the strip width W, i.e., $L = N_sW$.

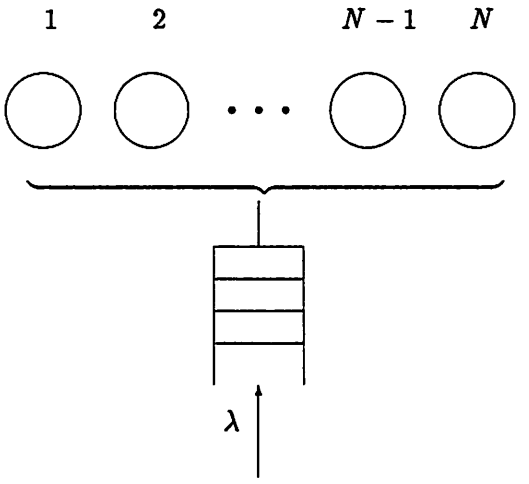


Figure 9: RAID 5 Queueing Model for Large I/O Applications

Let p_f be the probability that a write request is a full strip write, then

$$p_f = \sum_{m=1}^{\infty} P\{L = mW\}$$

which is equivalent to

$$p_f = P\{T_s = 0\}.$$

Let μ_f and μ_p be the service rates of full strip and partial strip writes, respectively. From (5), we have

$$\frac{1}{\mu_f} = 16.7\frac{W}{W+1} + \overline{N_s}\alpha.$$

In the following, we obtain the mean service time for partial strip writes, $1/\mu_p$. Suppose a partial strip write contains N_s strips and a tail of T_s blocks. We consider two strategies for calculating the new parity block for the partial strip (see Section 2.1.2).

Calculation of $1/\mu_p$ for the RT strategy :

The strategy *RT* reads the T_s old tail blocks and the parity block, *XOR*s them with the new tail blocks, and then writes them back to the disks. The time required to calculate the new parity block is negligible compared to the rotation and transfer times, and is ignored in our analysis. Therefore, the rotation and transfer time for such a write is

$$X = \max_{j=1,T_s+1}\{R_j\} + (N_s+1)\alpha + \Delta$$

where $\Delta = 16.7 + \alpha$, as defined in the previous subsection. We have

$$E[X|L=k] = (m+1)\alpha + \Delta + 16.7\frac{n+1}{n+2}$$

where $k = mW + n$.

Thus,

$$E[X] = (\overline{N_s} + 1)\alpha + \Delta + \frac{16.7}{W-1} \sum_{n=1}^{W-1} \frac{n+1}{n+2} \tag{9}$$

and

$$\frac{1}{\mu_p} = E[S] + E[X]. \tag{10}$$

Calculation of $1/\mu_p$ for the RC Strategy :

The second strategy, RC, reads the complementary blocks of the tail to calculate the new parity block. Let disk i be the disk which contains the parity block for the partial strip. We distinguish between the two possible events described in Section 2.1.2, according to whether the parity block has or has not been calculated by the time t' when disk i is ready to write it (Figure 5). Label these two events as A and B respectively and p_A and p_B as the probabilities associated with them, $p_B = 1 - p_A$. If event A occurs, then the service ends when the last of the set of disks writing the tail blocks and disk i complete. If event B occurs, then disk i will perform an additional full rotation after which it will write the parity block. The service ends at that point in time.

Assume that the partial strip write has a n block tail ($0 < n < W$), then $A = \{R_i \geq \tilde{R}\}$ and $B = \{R_i < \tilde{R}\}$ where $\tilde{R} = \max_{j=n+1,W}\{R_j + \alpha\}$ having density function

$$\begin{aligned} f_{\tilde{R}}(y) &= \frac{d}{dx}\left(\frac{x-\alpha}{16.7}\right)^{W-n} \\ &= \frac{W-n}{(16.7)^{W-n}}(x-\alpha)^{W-n-1}. \end{aligned}$$

Consequently,

$$\begin{aligned} E[X|L=k] &= p_A E[X|L=k,A] + p_B E[X|L=k,B] \\ &= p_A\left(E[R|L=k,A] + E[T|L=k,A]\right) + p_B\left(E[R|L=k,B] + E[T|L=k,B]\right) \end{aligned} \tag{11}$$

where $k = mW + n$.

We have

$$\begin{aligned} p_A &= \int_\alpha^{16.7} P\{R_i \geq y\} f_{\tilde{R}}(y)\, dy \\ &= \frac{W-n}{(16.7)^{W-n}} \int_\alpha^{16.7} \left(1 - \frac{y}{16.7}\right)(y-\alpha)^{W-n-1}\, dy \\ &= \frac{1}{W-n+1}\left(1 - \frac{\alpha}{16.7}\right)^{W-n+1}. \end{aligned} \tag{12}$$

The mean transfer time (including the additional full rotation when required) conditioned on events A and B are (see Figure 5 (a) and (b))

$$E[T|L=k,B] = m\alpha + \Delta \tag{13}$$

$$E[T|L=k,A] = m\alpha + \alpha \tag{14}$$

where $\Delta = 16.7 + \alpha$ as described in the last subsection.

We focus now on the two conditional rotational latencies. First,

$$p_B E[R|L=k,B] = p_B E[R_i|L=k,B]$$

and

$$\begin{aligned} p_B P\{R_i < x | L = k, R_i < \tilde{R}\} &= P\{R_i < x, R_i < \tilde{R} | L = k\} \\ &= \int_{\alpha}^{16.7+\alpha} P\{R_i < x, R_i < y | L = k\} f_{\tilde{R}}(y)\, dy \\ &= \begin{cases} \frac{x}{16.7} & 0 < x \le \alpha, \\ \frac{1}{16.7}\int_{\alpha}^{x} y f_{\tilde{R}}(y)\, dy + \frac{x}{16.7}\int_{x}^{16.7+\alpha} f_{\tilde{R}}(y)\, dy & \alpha < x < 16.7. \end{cases} \end{aligned}$$

Hence,

$$\begin{aligned} p_B E[R_i | L = k, B] &= \int_0^{16.7} x\, d_x P\{R_i < x, R_i < \tilde{R} | L = k\} \\ &= \frac{1}{16.7}\int_0^{16.7} x\, dx - \frac{1}{(16.7)^{W-n+1}}\int_{\alpha}^{16.7} x(x-\alpha)^{W-n}\, dx \\ &= 8.3 - \frac{16.7}{W-n+2}\left(1 - \frac{\alpha}{16.7}\right)^{W-n+2} - \frac{\alpha}{W-n+1}\left(1 - \frac{\alpha}{16.7}\right)^{W-n+1}. \end{aligned} \tag{15}$$

Second,

$$p_A E[R | L = k, A] = p_A E\left[\max\left\{R_i, \max_{j=1,n}\{R_j\}\right\} | L = k, A\right]$$

and

$$\begin{aligned} & p_A P\left\{\max\left\{R_i, \max_{j=1,n}\{R_j\}\right\} < x | L = k, R_i \ge \tilde{R}\right\} \\ &= P\left\{\max_{j=1,n}\{R_j\} < x | L = k\right\} \cdot P\left\{\tilde{R} \le R_i < x | L = k\right\}. \end{aligned}$$

Since

$$P\left\{\max_{j=1,n}\{R_j\} < x | L = k\right\} = \left(\frac{x}{16.7}\right)^n$$

and

$$\begin{aligned} P\left\{\tilde{R} \le R_i < x | L = k\right\} &= \int_{\alpha}^{16.7} P\{y \le R_i < x | L = k\} f_{\tilde{R}}(y)\, dy \\ &= \int_{\alpha}^{16.7} \frac{x-y}{16.7} f_{\tilde{R}}(y)\, dy \\ &= \left(1 - \frac{\alpha}{16.7}\right)^{W-n}\left(\frac{x-\beta}{16.7}\right), \qquad x \ge \beta \end{aligned}$$

where $\beta = \alpha + \frac{W-n}{W-n+1}(16.7 - \alpha)$, we have

$$\begin{aligned} & p_A E\left[\max\left\{R_i, \max_{j=1,n}\{R_j\}\right\} | L = k, A\right] \\ &= \int_{\beta}^{16.7} x\, d_x\left[\left(\frac{x}{16.7}\right)^n\left(\frac{x-\beta}{16.7}\right)\left(1 - \frac{\alpha}{16.7}\right)^{W-n}\right] \\ &= \left(1 - \frac{\alpha}{16.7}\right)^{W-n}\left[16.7\frac{n+1}{n+2}\left(1 - \left(\frac{\beta}{16.7}\right)^{n+2}\right) - \beta\frac{n}{n+1}\left(1 - \left(\frac{\beta}{16.7}\right)^{n+2}\right)\right]. \end{aligned} \tag{16}$$

Substituting (12), (13), (14), (15), and (16) into (11) yields the conditional mean $E[X|L = k]$. Removal of the conditioning yields $E[X]$, and $1/\mu_p$ can be calculated as in (10).

Finally, the density function for the disk service time, Y, is given by,

$$f_Y(x) = p_r \mu e^{-\mu x} + p_w p_f \mu_f e^{-\mu_f x} + p_w (1 - p_f) \mu_p e^{-\mu_p x}.$$

Analogous to the discussion in the last subsection, the mean I/O response time is obtained in a similar way.

5 Performance Comparisons of RAID 1+ and RAID 5

In this section, we compare the performance of the RAID 1+ architecture and the RAID 5 architecture. The numerical results are obtained by solving the queueing models introduced in Section 3 and 4. I/O requests are assumed to arrive at the disk I/O subsystem according to a Poisson process with rate λ. The disk access time, as we discussed before, varies depending on the RAID architecture and the size of each request. The performance metric of greatest interest is the mean I/O response time. We also compare the maximum throughputs under RAID 1+ and RAID 5. To validate our analytic models, a series of simulations has been conducted. As a consequence, although there are small quantitative differences between the predictions made by our analytic models and simulations, the qualitative differences of RAID 1+ and RAID 5 remain unchanged.

5.1 Performance with Small I/O Applications

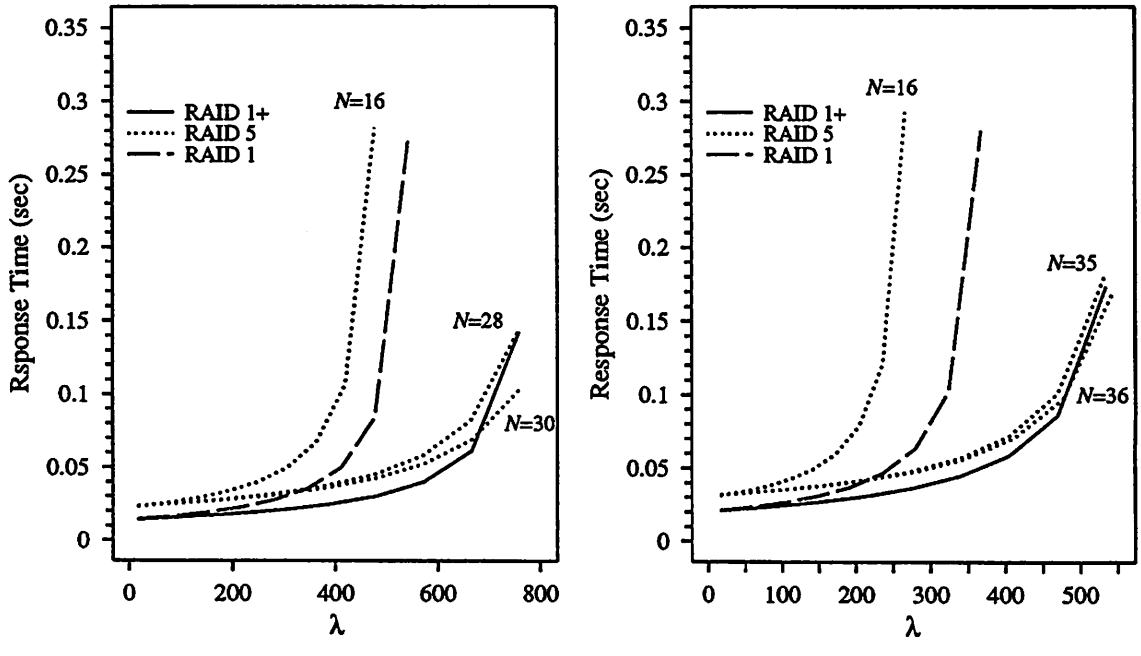


Figure 10: Performance of RAID 1+ and RAID 5 for Small I/O Applications. ($N = 16$)

For small I/O applications, we assume that each I/O request accesses 1 block consisting of 4096 bytes of data and is routed to each disk with equal probability. The number of disks in the array is set to $N = 16$ for both RAID 1+ and RAID 5. The analytic results are plotted in Figure 10, which shows mean I/O response time as a function of total arrival rate to the I/O subsystem. Two

cases are identified, one where most I/O requests are reads ($p_r = 0.75$) and the other where most requests are writes ($p_r = 0.25$). In both cases, RAID 1+ is observed to yield significantly better performance than RAID 5, especially as the write probability increases.

In fact, when all requests are reads ($p_r = 1$), both RAID 1+ and RAID 5 can serve up to N requests concurrently. However, RAID 1+ benefits by allowing a read to enter the shortest queue associated with one of its target disks. Furthermore, when system load is low, RAID 1+ gains from sending a request to the disk with the shortest seek distance if it finds that both target disks are available. For update operations under RAID 1+, a write has to be performed on two disks and its response time is the maximum of the two. Under RAID 5, however, in addition to accessing two disks (for data and parity respectively), a write must follow the read-update procedure. Hence, RAID 5 suffers an additional full rotational latency plus a transfer time for serving each write request. That is why RAID 5 performs worse relative to RAID 1+ as p_w increases.

We also plot the performance of RAID 1 (with MSP). As shown in Figure 10, with the same number of disks, RAID 1 also outperforms RAID 5. This result is consistent with that of [5]. However, the performance of RAID 1 is observed to be significantly worse than RAID 1+.

Now, we ask ourselves the question: how many disks are required by RAID 5 in order to achieve a performance comparable to that of RAID 1+? Increasing the number of disks may reduce the arrival rate to individual disk in the array, given the same total arrivals to the I/O subsystem. From Figure 10, we observe that when the read probability $p_r = 0.75$, the number of disks has to be nearly doubled. and when $p_r = 0.25$, more than twice as many disks are required by RAID 5. All of these results indicate that in order to provide reliable service for small I/O applications, RAID 5 suffers considerably due to the high write cost, and therefore exhibits a much worse performance than RAID 1+.

5.2 Performance with Large I/O Applications

In the large I/O application scenario, we consider a disk array of 64 disks. If a workstation disk has a capacity of 500M bytes, a disk array consisting of 64 such disks can yield 32G bytes of raw storage capacity, which provides ample capacity to replace mainframe disks.

To provide workload for large I/O applications, we assume that the I/O request size L to be a *r.v.* with mean $\overline{L} = 256$ (1M bytes). Since the performance of RAID 5 is significantly affected by the *full strip I/O* probability p_f, we select p_f as an adjustable parameter, and begin with a discussion of RAID 5. Specifically, L is expressed as $L = N_s W + T_s$, where $W = N - 1$ for RAID 5, N_s and T_s are independent *r.v.*'s, N_s is arbitrarily distributed, and the tail T_s is distributed as follows:

$$P\{T_s = n\} = \begin{cases} p_f & n = 0, \\ (1 - p_f)\frac{1}{N-2} & n = 1, 2, \cdots, N-2. \end{cases} \tag{17}$$

Hence, if we express integer k as $k = m(N-1) + n$, we have

$$P\{L = k\} = \begin{cases} P\{N_s = m\}p_f & n = 0, \\ P\{N_s = m\}(1 - p_f)\frac{1}{N-2} & n = 1, 2, \cdots, N-2. \end{cases} \tag{18}$$

Detailed calculations for obtaining the mean read service time $1/\mu$ for RAID 5, given the distribution of L as described above, are given in Appendix B.

In order to compare the performance of RAID 1+ and RAID 5, we use the same workload defined in (18) for the RAID 1+ architecture. When treated in this way, a full strip I/O operation under RAID 5 becomes a partial strip I/O under RAID 1+, since the strip width of RAID 1+, $N/2$, normally does not divide $N - 1$. Therefore, the full strip probability p_f introduced above only makes sense to RAID 5. This may lead to a slight penalty to the RAID 1+ architecture.

The numerical results, showing in Figure 11, are obtained by assuming the N_s to be geometrically distributed with mean $1/q = (\overline{L} - \overline{T_s})/(N-1)$, using the model described in Section 3.2. As there is little difference in the performance of RAID 1+ for $p_f = 1$ and 0.5, only the performance for $p_f = 0.5$ is shown.

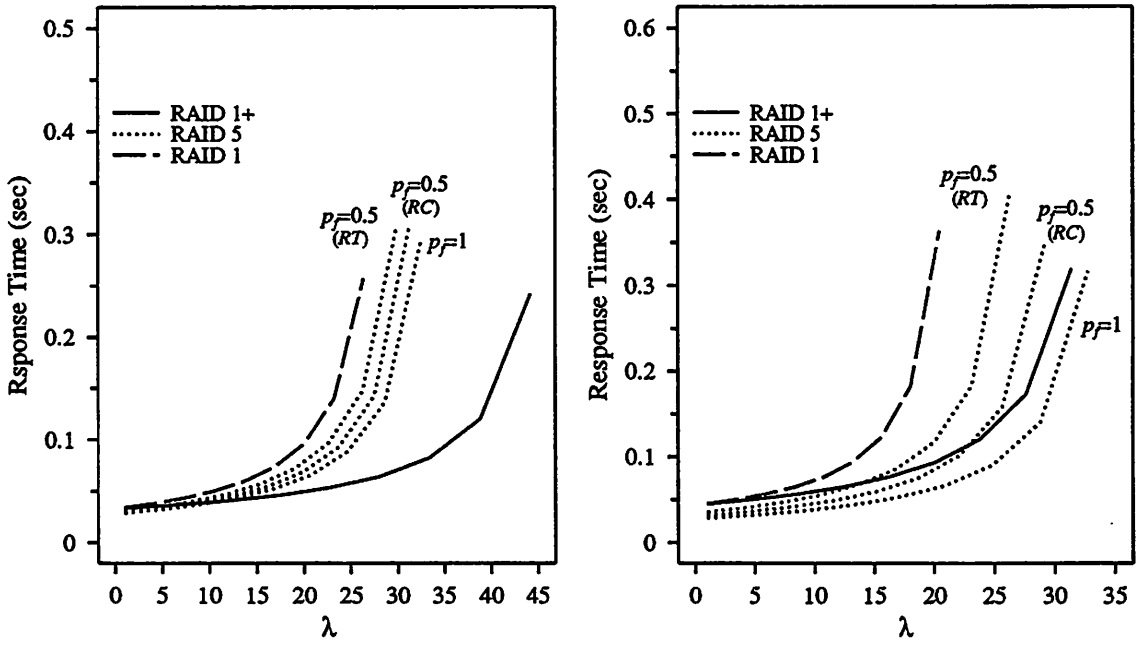


Figure 11: Performance of RAID 1+ and RAID 5 for Large I/O Applications. ($N = 64$)

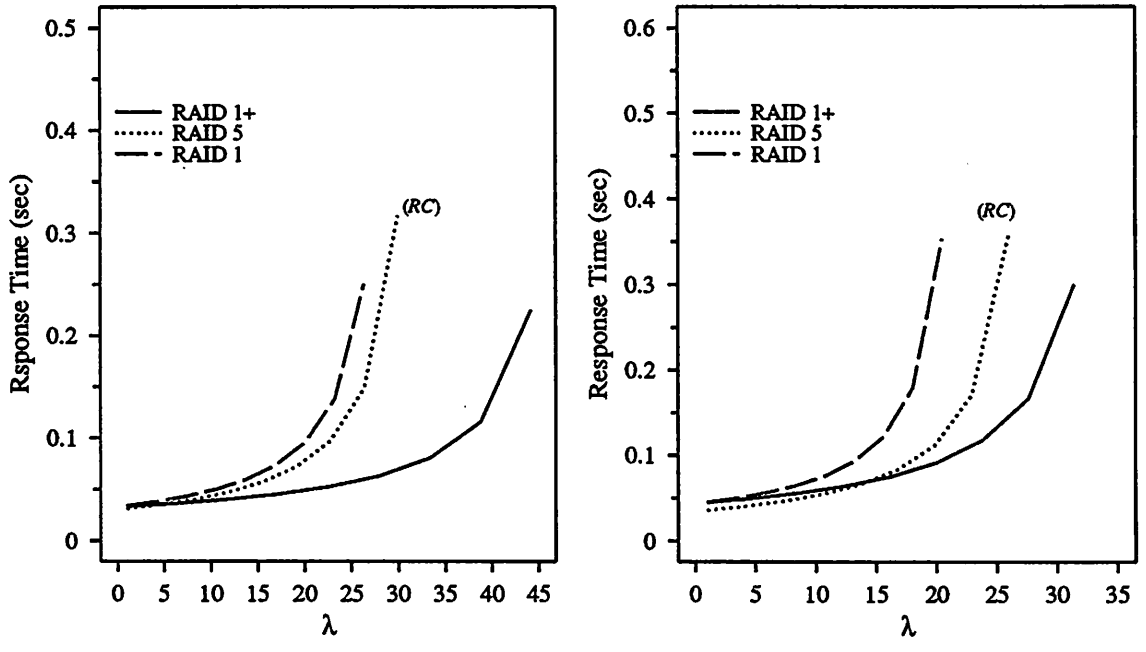
Figure 11 illustrates the performance of RAID 1+ and RAID 5 for different values of the *full strip probability* p_f, and the two strategies to calculate parity blocks for RAID 5. We observe from Figure 11(a), that RAID 1+ outperforms RAID 5 when most of the requests are reads. This is because RAID 1+ 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 11(b), the performance of RAID 5 degenerates rapidly as p_f decreases, because of the additional disk service time required for the read-update procedure of partial strip writes. When half of the writes are to a partial strip, RAID 1+ is observed to outperform RAID 5 at high loads.

Figure 11 also shows that, in the case of RAID 5, strategy RC, which reads complementary blocks of the tail for calculating new parity blocks, outperforms the strategy RT, which reads the tail, for the reasons discussed in Section 4.2.

Finally, consistent with the results obtained in [5], RAID 1 (with MSP) exhibits a worse performance than RAID 5 in large I/O applications, because it experiences a longer transfer time but does not support concurrent reads.

In the above discussions, we assumed that most of the I/O requests to RAID 5 are to perform

full strip I/O operations ($p_f \geq 0.5$). This requires the file system or buffer manager to be aware of the strip width of the underlying disk arrays. Whenever the number of disks in the array increases or decreases, the file system or buffer manager has to adapt to the change in order to maintain high performance. Now we compare the performance of the RAID 1+ and RAID 5 architectures when neither the file system nor the buffer manager know the strip width of the array. In particular, we assume the request size L to be uniformly distributed in [64, 448], such that each request access at least one strip of data and the mean I/O size $\overline{L} = 256$. This corresponds to request size ranging roughly from 256K to 2M bytes. The results are shown in Figure 12. Compared to Figure 11, we observe that the results for RAID 1+ are slightly better here, but the performance of RAID 5 degrades dramatically.



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

Figure 12: Request Size Uniformly Distributed for Large I/O Applications. ($N = 64$)

5.3 The Maximum Throughput of RAID 1+ and RAID 5

While the number of disks, N, and the mean I/O size were fixed in our previous discussions, we vary them here to illustrate the maximum I/O subsystem throughput as a function of these two parameters. Again, because the performance of RAID 1+ is insensitive to the value of p_f, only the performance for $p_f = 0.5$ is shown.

- **Maximum Throughput vs. Number of Disks**
  In this study, we take the mean I/O size to be 1M bytes for large I/O applications. The number of disks in the array varies from 16 to 128. Figure 13(a) shows that RAID 1+ achieves almost 37% higher throughput when most of the requests are reads. This is due to the fact that concurrent reads are supported by RAID 1+. On the other hand, when $p_r = 0.25$, we observe from Figure 13(b) that RAID 5 yields a higher throughput when the

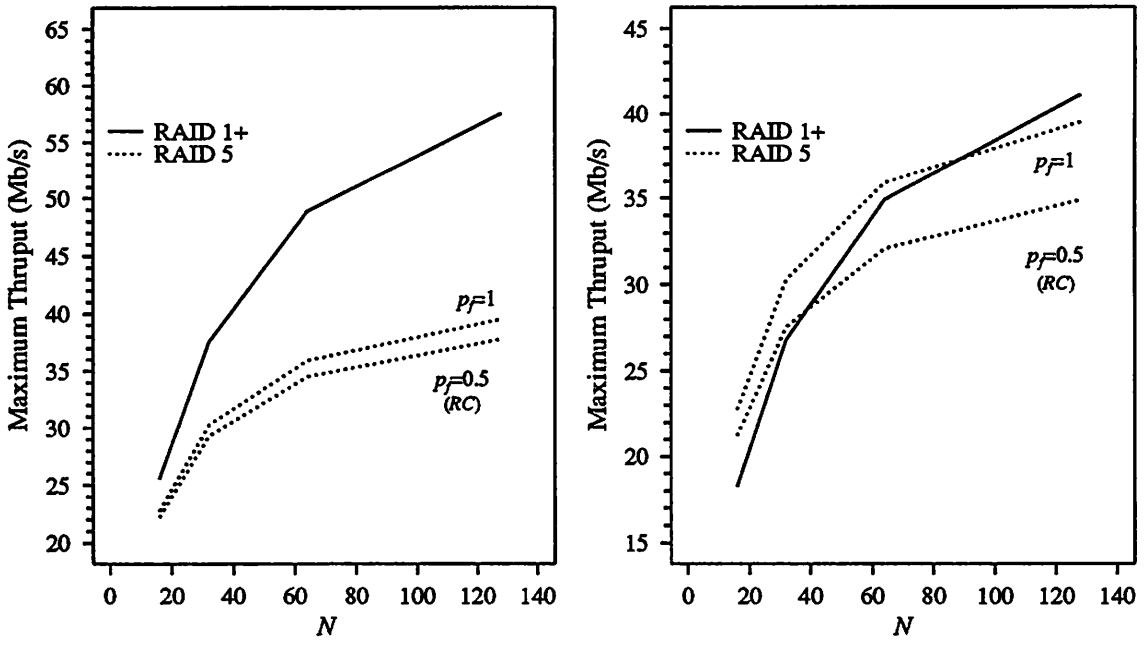


Figure 13: Varying Number of Disks in a Array (Large I/O).

number of disks used is small, since in this case, more data has to be transferred from each disk and therefore the transfer time dominates the disk access time. As the number of disks increases, this dominance decreases. As a result, when half of the writes are to a partial strip, RAID 1+ outperforms RAID 5 soon after the number of disks exceeds 40.

- **Maximum Throughput vs. I/O Request Size**
  Figure 14 illustrates the maximum throughput of RAID 1+ and RAID 5 as a function of mean I/O request size. The number of disks is fixed to 16. Thus, we vary the mean request size from 16 to 256 blocks, which corresponds to 64K to 1M bytes for each I/O. For the same reason as above, when $p_r = 0.75$, RAID 1+ achieves higher throughput because of the concurrent reads. When the majority of the requests perform updates and $p_f = 0.5$, RAID 5 begins to show higher throughput after the I/O size exceeds 125 blocks, which corresponds to each I/O request accessing more than 500K bytes on the average from a 16 disk array.

5.4 Sensitivity to the Sequential Access Probability

In the preceding studies, the sequential access probability p_s was selected to be 0.5. In this subsection we investigate the effect that varying p_s has on the performance of the RAID 1+ and RAID 5 architectures.

The results are illustrated in Figure 15 for both small and large I/O applications, where the device utilizations are fixed to 0.5 and 0.8, respectively. For large I/O applications, as before, only the performance of RAID 1+ is shown for $p_f = 0.5$. As expected, the I/O response time decreases for both RAID 1+ and RAID 5 as p_s increases. However, the differences in mean response times of these two architectures remain the same in all cases.

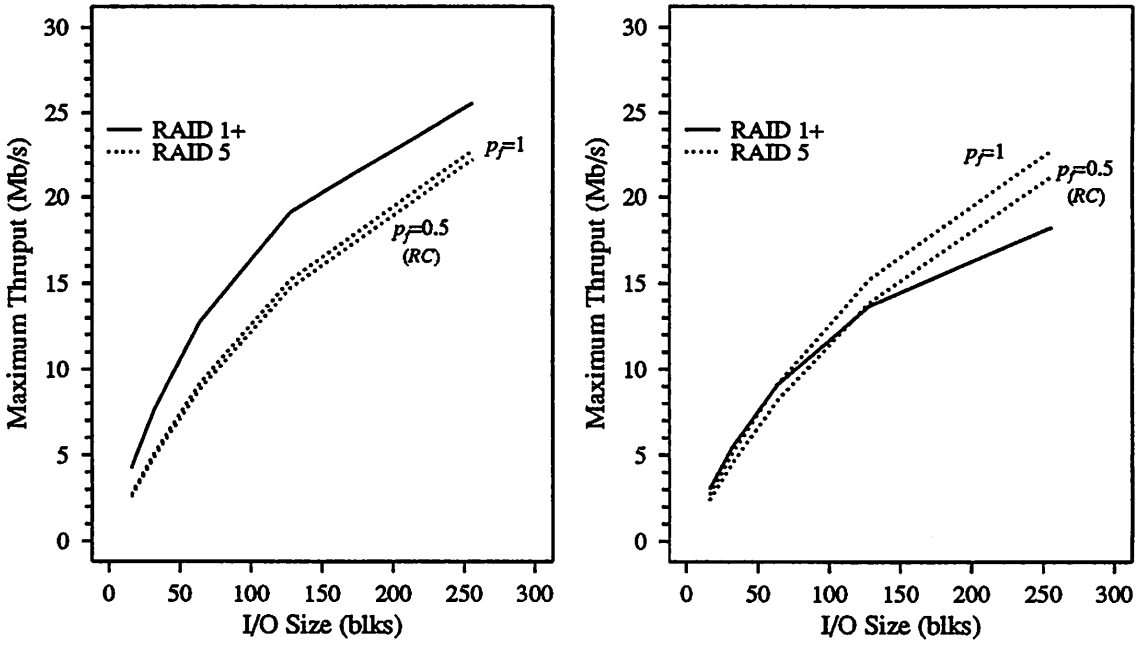


Figure 14: Varying Mean I/O Size ($N = 16$, Large I/O).

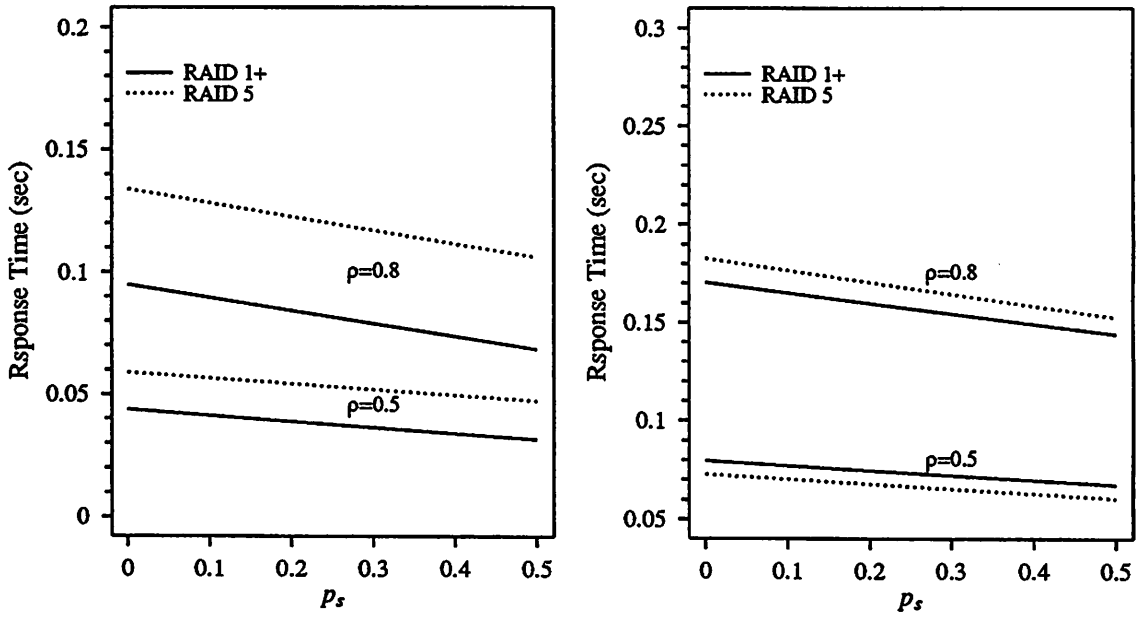
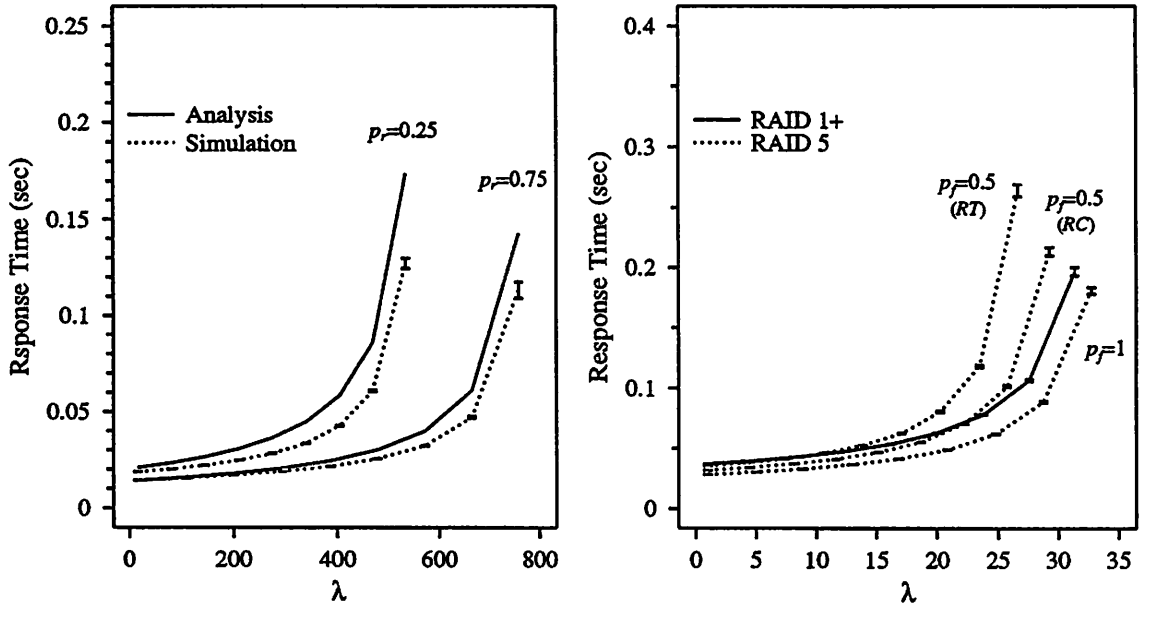


Figure 15: Vary the Sequential Access Probability p_s with Fixed Device Utilizations.

5.5 Simulation Experiments

In our analytical models, disk service times are assumed to be exponentially distributed for tractability. However, in reality, disk service time are not exponential. In order to validate our performance results on both RAID 1+ and RAID 5 architectures, we developed a simulator for the RAID architectures that simulates the seek, rotational latency, and transfer times on an individual disk properly. The rotational latency of each disk is uniformly distributed in $[0, 16.7]$ and the transfer time is determined by the disk transfer rate and the I/O size. For large I/O applications, the number of strips in a request, N_s, is selected from a geometric distribution by the simulator. The simulation of RAID 5 does not account for the synchronization delay that is incurred while updating the parity block during the execution of a write request in the small I/O scenario. Hence, the predictions made for RAID 5 remain optimistic.

The simulation results are obtained by averaging over 40 runs, each run including 50,000 I/O requests. We obtained 95% confidence intervals whose widths are less than 2% of the estimated point values.



(a) Analysis vs. Simulations (*RAID 1+ for Small I/O*)

(b) Simulation Results (p_r= 0.25, *Large I/O*)

Figure 16: Model Validation via Simulations.

As shown in Figure 16(a), the analytic and simulation results do not match quite well, because of the unrealistic exponential service time assumption made in our analysis. However, the qualitative differences between RAID 1+ and RAID 5 are the same in all cases. Figure 16(b) shows the simulation results for the case of read probability $p_r = 0.25$ in large I/O applications. The relative position of all the curves are exactly the same for RAID 1+ and RAID 5 as illustrated in Figure 11(b). We have observed the same behavior in a number of other experiments which are not reported here due to lack of space.

6 Summary

In this paper, we studied the performance of two different disk array architectures, a *mirrored array* (RAID 1+) and *rotated parity array* (RAID 5). In particular, we presented approximate analytic queueing models for RAID 1+ and RAID 5, respectively. Both of these disk array architectures are evaluated under two typical application scenarios. On one hand, in transaction processing applications or UNIX environments, the disk access pattern is characterized by many small I/O requests, each randomly accessing a portion of a database or a file system. On the other hand, in supercomputing or image processing applications, the disk access pattern consists of fewer I/O requests, but each accessing a large amount of data.

The main difference between this study and previous studies is that a more efficient policy is used for the RAID 1+ architecture. Consequently, our analytic results show that the RAID 1+ architecture outperforms RAID 5 in terms of minimizing the average I/O response time, especially for small I/O applications. The only exception occurs when the I/O size is very large, most requests are writes, and most of the writes perform a full strip write operation. In addition, we also studied the maximum I/O subsystem throughput under both RAID 1+ and RAID 5. As a consequence, RAID 5 shows a higher throughput only when most of the I/O requests are writes, the I/O size is large, and fewer disks are included in the array. Finally, simulations were conducted to validate our analytic models.

As mentioned before, RAID 1+ gains a performance benefit but loses half of its capacity, as compared to only $1/N$ loss under RAID 5. A trade-off might be to provide duplicate copies only for heavily used data, such as some system files. In this case, many interesting issues, such as how to layout various files, select optimal strip width, as well as design proper scheduling policies, are still open. Another issue of interest is the performance of RAID architectures during failure. As the number of disks in the array increases, the MTTF decreases fast. Consequently, the disk array will face a disk failure and be required to operate while recovery is in progress. Thus, one of our future research interests will be to evaluate the performance of different RAID architectures during failure under various recovery techniques.

Appendix A: Calculations for the RAID 1+ Architecture

In this Appendix, we present a solution to the Markov chain $\mathbf{N}(t) = (N_{max}(t), N_{min}(t), N_3(t))$, which provides a lower bound for the RAID 1+ architecture for small I/O applications, as described in Section 3.1.

Two variations of the SQP-MS scheduler have been identified in the context (Section 2.1.1), which differ from each other by routing those reads that find both disks busy and the two queue lengths to be the same at the moment of their arrival. In the following, we first present an analytic model for the *intelligent scheduler (IS)*, and then simply describe modifications required to model the *simple scheduler (SS)*. The numerical results for the two variations, as expected, are very close to each other such that the performance differences can be perceived by only tens of nanoseconds on the mean I/O response times, with IS slightly better than SS.

Model for the Intelligent Scheduler :

In this model, the state variables N_{max} and N_{min} denote the longest and the shortest queue lengths at time t, respectively. $N_3(t)$ indicates whether the most recent busy period was initiated by a read and this read is still in service ($N_3(t) = 1$), or this read has already finished

or the busy period was initiated by a write ($N_3(t) = 0$). Observe that when $N_3(t) = 1$, the read that initiated the most recent busy period (which took a fast service rate μ') is always in the longest queue at time t because of the behavior of the *intelligent scheduler*.

To solve this model, we first calculate the stationary probability distribution $P\{\mathbf{N} = (m,n,l)\} = q(m,n,l)$, $m = 0,1,\cdots$; $n = 0,1,\cdots,B$; $l = 0,1$, and then obtain the mean I/O response times. We define the steady state probability vector

$$Y = (y(0), y(1), y(2), \cdots)$$

where $y(0)$ is a $2B+1$ element vector $y(0) = [q(0,0,0), q(1,1,0), q(1,1,1), \cdots, q(B,B,0), q(B,B,1)]$, and $y(i)$ is a $2B+2$ element vector $y(i) = [q(i,0,0), q(i,0,1), \cdots, q(i+B,B,0), q(i+B,B,1)]$, $i = 1,2,\cdots$. These probabilities satisfy the following equations,

$$\begin{aligned}
\lambda q(0,0,0) &= \mu q(1,0,0) + \mu' q(1,0,1), \\
(\lambda+\mu) q(1,0,0) &= \mu q(2,0,0) + 2\mu q(1,1,0) \\
&\quad +\mu'[q(2,0,1) + q(1,1,1)], \\
(\lambda+\mu') q(1,0,1) &= p_r \lambda q(0,0,0) + \mu q(1,1,1), \\
(\lambda+\mu) q(i,0,0) &= \mu[q(i+1,0,0) + q(i,1,0)] + \mu' q(i+1,0,1), \\
(\lambda+\mu') q(i,0,1) &= \mu q(i,1,1), && i = 2,3\cdots \\
(\lambda+2\mu) q(i,i,0) &= p_r \lambda q(i,i-1,0) + p_w \lambda q(i-1,i-1,0) \\
&\quad \mu q(i+1,i,0) + \mu' q(i+1,i,1), && i = 1,\cdots,B. \\
(\lambda+\mu+\mu') q(1,1,1) &= p_r \lambda q(1,0,1), \\
(\lambda+\mu+\mu') q(i,i,1) &= p_r \lambda q(i,i-1,1) + p_w \lambda q(i-1,i-1,1), && i = 2,\cdots,B. \\
(\lambda+2\mu) q(i+1,i,0) &= p_r \lambda[q(i+1,i-1,0) + q(i,i,0)] \\
&\quad +p_w \lambda q(i,i-1,0) \\
&\quad +2\mu q(i+1,i+1,0) + \mu q(i+2,i,0) \\
&\quad +\mu'[q(i+1,i+1,1) + q(i+2,i,1)], \\
(\lambda+\mu+\mu') q(i+1,i,1) &= p_r \lambda[q(i+1,i-1,1) + q(i,i,1)] \\
&\quad +p_w \lambda q(i,i-1,1) + \mu q(i+1,i+1,1), && i = 1,\cdots,B-1. \\
(p_w\lambda+2\mu) q(B+1,B,0) &= p_r \lambda q(B+1,B-1,0) + p_w \lambda q(B,B-1,0) \\
&\quad +\lambda q(B,B,0) + \mu q(B+2,B,0) \\
&\quad +\mu' q(B+2,B,1), \\
(p_w\lambda+\mu+\mu') q(B+1,B,1) &= p_r \lambda q(B+1,B-1,1) + p_w \lambda q(B,B-1,1) \\
&\quad +\lambda q(B,B,1), \\
(\lambda+2\mu) q(i,j,0) &= p_r \lambda q(i,j-1,0) + p_w \lambda q(i-1,j-1,0) \\
&\quad +\mu[q(i+1,j,0) + q(i,j+1,0)] \\
&\quad +\mu' q(i+1,j,1), \\
(\lambda+\mu+\mu') q(i,j,1) &= p_r \lambda q(i,j-1,1) + p_w \lambda q(i-1,j-1,1) && i = 3,4,\cdots \\
&\quad +\mu q(i,j+1,1), && j = 1,\cdots,\min(i-2,B-1) \\
(p_w\lambda+2\mu) q(i,B,0) &= p_w \lambda[q(i-1,B-1,0) + q(i-1,B,0)] \\
&\quad +p_r \lambda q(i,B-1,0) + \mu q(i+1,B,0) \\
&\quad +\mu' q(i+1,B,1), \\
(p_w\lambda+\mu+\mu') q(i,B,1) &= p_w \lambda[q(i-1,B-1,1) + q(i-1,B,1)] \\
&\quad +p_r \lambda q(i,B-1,1), && i = B+2, B+3, \cdots.
\end{aligned}$$

The infinitesimal generator Q satisfying $YQ = 0$ is listed below,

$$Q = \begin{bmatrix} B_1 & B_0 & 0 & 0 & 0 & \cdots \\ B_2 & A_1 & A_0 & 0 & 0 & \cdots \\ 0 & A_2 & A_1 & A_0 & 0 & \cdots \\ 0 & 0 & A_2 & A_1 & A_0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

where B_1 is a $2B+1$ dimensional matrix,

$$B_1 = \begin{bmatrix} -\lambda_i & \lambda_i p_w & 0 & & & & \\ & -(\lambda_i + 2\mu) & 0 & \lambda_i p_w & & & \\ & & -(\lambda_i + \mu + \mu') & 0 & \ddots & & \\ & & & -(\lambda_i + 2\mu) & \ddots & \lambda_i p_w & \\ & & & & \ddots & 0 & \\ & & & & & -(\lambda_i + \mu + \mu') \end{bmatrix}$$

B_2 is a $(2B+2) \times (2B+1)$ matrix,

$$B_2 = \begin{bmatrix} \mu & \lambda_i p_r & & & & & & \\ \mu' & 0 & \lambda_i p_r & & & & & \\ 0 & \mu & 0 & \lambda_i p_r & & & & \\ & \mu' & 0 & 0 & \lambda_i p_r & & & \\ & & 0 & \mu & 0 & \ddots & \lambda_i p_r & \\ & & & \mu' & 0 & \ddots & 0 & \lambda_i p_r \\ & & & & 0 & \ddots & \mu & 0 \\ & & & & & & \mu' & 0 \end{bmatrix}$$

B_0 is a $(2B+1) \times (2B+2)$ matrix,

$$B_0 = \begin{bmatrix} 0 & \lambda_i p_r & & & & & & & & \\ 2\mu & 0 & \lambda_i p_r & & & & & & & \\ \mu' & \mu & 0 & \lambda_i p_r & & & & & & \\ & 0 & 2\mu & 0 & \lambda_i p_r & & & & & \\ & & \mu' & \mu & 0 & \lambda_i p_r & & & & \\ & & & 0 & 2\mu & 0 & \ddots & & & \\ & & & & \mu' & \mu & \ddots & \lambda_i p_r & & \\ & & & & & 0 & \ddots & 0 & \lambda_i p_r & \\ & & & & & & \ddots & 2\mu & 0 & \lambda_i \\ & & & & & & & \mu' & \mu & 0 & \lambda_i \end{bmatrix}$$

and A_0, A_1, A_2 are $2B+2$ dimensional matrices,

$$A_0 = \begin{bmatrix} 0 & & & & & & \\ 0 & \ddots & & & & & \\ \mu & \ddots & 0 & & & & \\ & \ddots & 0 & 0 & & & \\ & & \mu & 0 & \lambda_i p_w & & \\ & & & \mu & 0 & \lambda_i p_w \end{bmatrix}$$

$$A_1 = \begin{bmatrix} -(\lambda_i+\mu) & 0 & \lambda_i p_w & & & & & \\ & -(\lambda_i+\mu') & 0 & \lambda_i p_w & & & & \\ & & -(\lambda_i+2\mu) & 0 & \lambda_i p_w & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & -(\lambda_i+\mu+\mu') & 0 & \lambda_i p_w & \\ & & & & & -(\lambda_i p_w+2\mu) & 0 & \\ & & & & & & -(\lambda_i p_w+\mu+\mu') \end{bmatrix}$$

$$A_2 = \begin{bmatrix} \mu & 0 & \lambda_i p_r & & & & & & \\ \mu' & 0 & 0 & \lambda_i p_r & & & & & \\ & 0 & \mu & 0 & \lambda_i p_r & & & & \\ & & \mu' & 0 & 0 & \lambda_i p_r & & & \\ & & & 0 & \mu & 0 & \ddots & & \\ & & & & \mu' & 0 & \ddots & \lambda_i p_r & \\ & & & & & 0 & \ddots & 0 & \lambda_i p_r \\ & & & & & & \ddots & \mu & 0 \\ & & & & & & & \mu' & 0 \end{bmatrix}.$$

The λ_i above is the arrival rate of requests that are routed to mirror i, $\lambda_i = \lambda/(N/2), i = 1, 2, ..., N/2$; μ is the disk service rate; and μ' is the service rate for those reads that find both data copies available at the moment of their arrival.

Define the matrix $A = A_0 + A_1 + A_2$. Neuts [18] showed that Q is positive recurrent if $\pi A_0 e < \pi A_2 e$, where π is the unique solution to $\pi A = 0$, $\pi e = 1$. Here e is a column vector containing $2B+2$ ones. Whenever Q is positive recurrent, the stationary probability vector Y satisfies the matrix-geometric form

$$y(i) = y(1)R^i, \quad i = 0, 1, \cdots$$

where R is the minimal solution of $A_0 + RA_1 + R^2 A_2 = 0$. The matrix R can be obtained iteratively using the following approach. Let $R(n)$ denote the value of R after n iterations.

$$\begin{aligned} R(0) &= 0, \\ R(n+1) &= -A_0 A_1^{-1} - R^2(n) A_2 A_1^{-1}, \quad n > 0. \end{aligned}$$

Finally, the vectors $y(0)$ and $y(1)$ are obtained by solving the following set of equations,

$$\begin{aligned} y(0)B_1 + y(1)B_2 &= 0 \\ y(0)B_0 + y(1)(A_1 + RA_2) &= 0 \\ y(0)e' + y(1)[I-R]^{-1}e &= 1 \end{aligned}$$

where e' is a column vector with $2B+1$ ones.

The mean I/O response times for read and write requests are obtained in a following way. Let $y'(0)$ be a $2B+2$ element vector, $y'(0) = (0, y(0))$. The mean queue length for the shortest queue is

$$\begin{aligned}\overline{N_{min}} &= \sum_{k=0}^{B} k[y'(0) + \sum_{i=1}^{\infty} y(i)]e_k \\ &= [y'(0) + y(1)(I-R)^{-1}] \sum_{k=0}^{B} ke_k\end{aligned}$$

where e_k is a $2B+2$ element vector with the $(2k+1)$-th and $(2k+2)$-th elements set to one, and all other elements zero.

The mean read response time is

$$\begin{aligned}\overline{Z_r} &= q(0,0,0)\frac{1}{\mu'} + \sum_{j=0}^{B}\sum_{i=j,i\neq 0}^{\infty}[q(i,j,0)+q(i,j,1)](j+1)\frac{1}{\mu} \\ &= (\overline{N_{min}}+1)\frac{1}{\mu} + q(0,0,0)(\frac{1}{\mu'} - \frac{1}{\mu}).\end{aligned}$$

To obtain the mean write response time $\overline{Z_w}$, we first define $T_{i,j}$ to be the mean response time for write requests which find the two queue lengths to be $i-1$ and $j-1$ at the moment of their arrival. Then $T_{i,j}$ can be obtained by solving the following recurrence:

$$T_{i,j} = \frac{1}{2\mu} + \frac{1}{2}T_{i-1,j} + \frac{1}{2}T_{i,j-1} \qquad i,j = 0,1,2,\cdots$$

with the boundary conditions $T_{0,j} = j/\mu, j = 0,1,\cdots$, and $T_{i,0} = i/\mu, i = 0,1,\cdots$. The first term in the recurrence corresponds the average delay until the first of the two disks completes. Since the disk that finishes first can be either one in a mirrored pair with same probability, the write request observes the system with one less task in that queue. Hence, its average delay in this case corresponds to the second and the third terms in the recurrence.

Let T_i be a $2B+2$ element vector, $T_i = (T_{i,1}, T_{i,1}, T_{i+1,2}, T_{i+1,2}, \cdots, T_{i+B,B+1}, T_{i+B,B+1})$. The mean write response time is given by

$$\begin{aligned}\overline{Z_w} &= q(0,0,0)T_{1,1} + \sum_{j=0}^{B}\sum_{i=j,i\neq 0}^{\infty}[q(i,j,0)+q(i,j,1)]T_{i+1,j+1} \\ &= y'(0)T_1 + y(1)\sum_{i=1}^{\infty} R^{i-1}T_{i+1}.\end{aligned}$$

In calculation, the infinite summation can be truncated to a finite number which is large enough, say 500, since the I/O queue length can rarely exceed 500. Finally, the overall mean I/O response time is

$$\overline{Z} = p_r\overline{Z_r} + p_w\overline{Z_w}.$$

Model for the Simple Scheduler :

To model the *simple scheduler*, it is necessary for $N_3(t)$ to take an additional value $N_3(t) = 2$, which carries the semantics that the most recent busy period prior to t was initiated by a read,

this read is still in service, and its associated queue is the shortest one at time t. $N_3(t) = 1$, as described above, has the same meaning except that the read that initiated the most recent busy period is now in the longest queue at t. $N_3(t) = 0$ has the same meaning as before. Thus, if the system is stationary and in a state $(k, k, 1)$, $k = 1, 2, \cdots, B$, i.e., the system was started by a read which is still in service and both queue lengths are equal, it transits to the states $(k+1, k, 1)$ or $(k+1, k, 2)$ with a equal rate $(0.5\lambda_i p_r)$ on the event of an arrival of a new read. Transitions to and from other states remain unchanged as in the *intelligent scheduler* model.

With these modifications, we can construct a new Markov chain, $\mathbf{N}(t) = (N_{max}(t), N_{min}(t), N_3(t))$, in a similar manner as for the *intelligent scheduler*. Last, the modified Markov chain may still be solved by using the matrix-geometric method in a similar way as above, and the details are omitted here.

Appendix B: An Example of Obtaining the $1/\mu$ for Large I/O Applications

In this Appendix, we calculate the mean read service time $1/\mu$ for RAID 5. The calculations of mean write service time, $1/\mu_f$ and $1/\mu_p$, for RAID 5 are discussed in Section 4.2. The mean disk service time for RAID 1+ is obtained in the same way, given the distributions of L as in Section 5.2. (Note: there is no difference between read and write service time for RAID 1+.)

As stated before, L can be uniquely expressed as $L = N_s W + T_s$, where W is the strip width ($W = N - 1$), N_s is the number of strips, and T_s is the number of blocks in the tail. As stated in Section 5.2, we assume that N_s has an arbitrary distribution, N_s and T_s are independent of each other, the tail T_s is zero with probability p_f and is uniform in $[1, W-1]$ when greater than zero, i.e.,

$$P\{T_s = n\} = \begin{cases} p_f & n = 0, \\ (1-p_f)\frac{1}{W-1} & n = 1, 2, \cdots, W-1. \end{cases} \tag{19}$$

Hence, if we express integer k as $k = mW + n$, we have

$$P\{L = k\} = \begin{cases} P\{N_s = m\}p_f & n = 0, \\ P\{N_s = m\}(1-p_f)\frac{1}{W-1} & n = 1, 2, \cdots, W-1. \end{cases} \tag{20}$$

When $0 < n < W$, from (5) we obtain

$$\begin{aligned} E[X|L=k] &= \frac{1}{(16.7)^W}\int_{\alpha}^{16.7}\left[Wx^{W-n}(x-\alpha)^n + n\alpha x^{W-n}(x-\alpha)^{n-1}\right]dx \\ &\quad + \frac{1}{(16.7)^n}\int_{16.7}^{16.7+\alpha} nx(x-\alpha)^{n-1}\,dx + m\alpha \\ &= \phi_n^{(1)} + \phi_n^{(2)} + \phi_n^{(3)} + m\alpha \\ &= \phi_n + m\alpha \end{aligned} \tag{21}$$

where

$$\phi_n^{(1)} = \frac{W}{(16.7)^W}\int_0^{16.7-\alpha} y^n(y+\alpha)^{W-n}\,dy$$

$$= \frac{W}{(16.7)^W} \sum_{k=0}^{W-n} \frac{\binom{W-n}{k}}{W+1-k} (16.7-\alpha)^{W+1-k} \alpha^k$$

$$\phi_n^{(2)} = \frac{n\alpha}{(16.7)^W} \int_0^{16.7-\alpha} y^{n-1}(y+\alpha)^{W-n}\, dy$$

$$= \frac{n\alpha}{(16.7)^W} \sum_{k=0}^{W-n} \frac{\binom{W-n}{k}}{W-k} (16.7-\alpha)^{W-k} \alpha^k$$

$$\phi_n^{(3)} = \frac{n}{(16.7)^n} \int_{16.7-\alpha}^{16.7} y^{n-1}(y+\alpha)\, dy$$

$$= \frac{n}{n+1} 16.7 \left[1 - \left(1 - \frac{\alpha}{16.7}\right)^{n+1}\right] + \alpha \left[1 - \left(1 - \frac{\alpha}{16.7}\right)^n\right].$$

Substituting (5), (20) and (21) to (6) yields

$$E[X] = \sum_{m=1}^{\infty} P\{N_s = m\} P\{T_s = 0\} \left(16.7 \frac{W}{W+1} + m\alpha\right) + \sum_{m=1}^{\infty} P\{N_s = m\} \sum_{n=1}^{W-1} P\{T_s = n\}(\phi_n + m\alpha)$$

$$= \alpha \overline{N_s} + p_f 16.7 \frac{W}{W+1} + (1-p_f) \frac{1}{W-1} \sum_{n=1}^{W-1} \phi_n. \tag{22}$$

The term $\overline{N_s}$ is obtained from the expression,

$$\overline{N_s} = \frac{\overline{L} - \overline{T_s}}{W}$$

$$= \frac{\overline{L}}{W} - \frac{1-p_f}{2}.$$

Acknowledgments:

The authors are gratefully acknowledge James F. Kurose for his contributions to this work, especially for the early discussions on the ideas of this study.

References

[1] C. G. Bell, "The Mini and Micro Industries," *IEEE Comp. Mag.*, Vol.17, pp.14-30, Oct. 1984.

[2] D. Bitton, "Arm Scheduling in Shadowed Disks," *Proc. ComCon 89*, pp.132-136, 1989.

[3] D. L. Bultman, "High Performance SCSI Using Parallel Drive Technology," *Proc. BUSCON Conf.*, Anaheim, CA, Feb. 1988.

[4] P. Chen, G. Gibson, R. H. Katz, and D. A. Patterson, "An Evaluation of Redundant Arrays of Disks using an Amdahl 5890," *Perf. Eval. Rev.*, Vol.18, No.1, pp.74-85, May 1990.

[5] P. Chen, and D. A. Patterson, "Maximize Performance in a Stripped Disk Array," *Proc. Computer Architecture,* pp.322-331, June 1990.

[6] G. Copeland and T. Keller, "A Comparison of High-Availability Media Algorithms," *Proc. ACM SIGMOD Conf.*, Portland, OR, May 1989.

[7] J. M. Harker, et al, "A Quater Century of Disk File Innovation," *IBM J. Res. Dev.*, Vol.25, pp.677-689, Sept. 1981.

[8] W. Joy, Presentation at ISSCC'85 Panel Session, Feb. 1985.

[9] R. H. Katz, G. Gibson, and D. A. Patterson, "Disk System Architectures for High Performance Computing," *Proc. of the IEEE,* Vol.77, No.12, pp.1842-1858, Dec. 1989.

[10] M. Y. Kim, "Synchronized Disk Interleaving," *IEEE Trans. Comp.,* Vol. C-35, pp.978-988, Nov. 1986.

[11] M. Y. Kim and A. N. Tantawi, "Asynchronized Disk Interleaving: Approximating Access Delays," *IEEE Trans. Comp.,* Vol.40, No.7, pp.801-810, July 1991.

[12] L. Kleinrock, *Queueing Systems* Vol. I, John Wiley & Sons, 1975, pp.187-190.

[13] E. K. Lee and R. H. Katz, "Performance Consequences of Parity Placement in Disk Arrays," *Proc 4th Int'l Conf. on Archi. Supp. for Prog. Lang. and OS*, April, 91, pp.190-199.

[14] M. Livny, S. Khoshafian, and H. Boral, "Multi-Disk Management Algorithm," *Proc. SIGMETRICS*, May 1987.

[15] W. C. Lynch, "Do Disk Arms Move?" *Perform. Eval. Rev.*, Vol.1, pp.3-16, Dec. 1972.

[16] R. R. Muntz and J. C. Liu, "Performance Analysis of Disk Arrays under Failure," *Proc. 16th VLDB Conf.*, Brisbane, Australia, 1990. pp.162-173.

[17] M. Nelson, B. Welch, and J. Ousterhout, "Caching in the Sprite Network File System," *ACM Trans. on Comp. Sys.,* Vol.6, No.1, Feb. 1988, pp.134-154.

[18] M. F. Neuts, *Matrix-Geometric Solutions in Stochastic Models - An Algorithmic Approach,* John Hopkins University Press, 1981.

[19] S. Ng, D. Lang, and R. Selinger, "Trade-offs Between Devices and Paths in Achieving Disk Interleaving," *Proc. 15th Int'l Sym. on Comp. Archit.*, Hawaii, May 1988.

[20] S. Ng, "Some Design Issues of Disk Array," *IEEE Spring ComCon,*, pp.137-142, 1989.

[21] M. Ogata and M. Flynn, "A Queueing Analysis for Disk Array Systems," *Stanford Tech. Rep. CSL-TR-90-443*, Aug. 1990.

[22] T. M. Olson, "Disk Array Performance in a Random I/O Environment," *Comput. Archit. News*, Vol.17, No.5, Sept. 1989, pp.71-77.

[23] J. K. Ousterhout and F. Douglis, "Beating the I/O Bottleneck: A Case for Log-structured File Systems," *UCB/CSD 88/467*, Computer Science Division (EECS), UC/Berkeley, CA, Oct. 1988.

[24] D. A. Patterson, G. Gibson, and R. H. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)," *Proc. ACM SIGMOD Conf.*, pp.109-116, July, 1988.

[25] D. A. Patterson, P. Chen, G. Gibson, and R. H. Katz, "Introduction to Redundant Arrays of Inexpensive Disks (RAID)," *IEEE Spring ComCon,*, pp.112-117, 1989.

[26] A. Poston, "A High Performance File System for UNIX," *USENIX Proc.*, Sept. 1988, pp.215-226.

[27] A. L. N. Reedy and P. Banerjee, "An Evaluation of Multiple-Disk I/O Systems," *IEEE Trans. Comp.*, Vol.38, No.12, Dec. 1989.

[28] K. Salem and H. Garcia-Molina, "Disk Striping," *Proc. IEEE Data Engineering Conf.*, Los Angeles, CA, Feb. 1986.

[29] M. Schroeder, D. Gifford, and R. Needham, "A Caching File System for a Programmer's Workstation," *Proc. 10th Sym. on Operating System Principles,* Dec. 1985, pp.25-34.

[30] M. Schulze, G. Gibson, R. Katz, D. Patterson, "How Reliable is a RAID?" *Proc. IEEE Spring ComCon Conf,* San Francisco, CA, Feb. 1989.

[31] R. A. Scranton and D. A. Thompson, "The Access Time Myth," *IBM Tech Report RC-10197*, pp.1-8, Sept. 1984.

[32] M. Stonebraker and G. A. Schloss, "Distributed RAID - a New Multiple Copy Algorithm," *Proc. 6th Int'l. Conf. on Data Engineering*, Feb., 1990.

[33] D. Towsley, S. Chen, and S-P. Yu, "Performance Analysis of a Fault tolerant Mirrored Disk System," *Proc. PERFORMANCE'90,* Sept. 1990.

[34] D. Towsley and S. Chen, "Design and Modeling Scheduling Policies for Two Server Fork/Join Queueing Systems," *submitted to the Performance Evaluation Journal*, March, 1991.