PERFORMANCE ANALYSIS OF A FAULT TOLERANT MIRRORED DISK SYSTEM

D. Towsley, S. Chen, S-P. Yu
Department of Computer and Information Science
University of Massachusetts
Amherst, MA 01003

COINS Technical Report 91-14

Performance Analysis of a Fault Tolerant Mirrored Disk System *

Don Towsley, Shenze Chen , and Shou-Pin Yu †

Department of Computer & Information Science
University of Massachusetts
Amherst, MA 01003
U.S.A.

Abstract

Disk mirroring has found widespread use as a mechanism for providing fault tolerance in highly reliable computer systems. In this paper we explore the performance of seven different policies for scheduling reads and updates on such a system containing two disks. These policies differ from each other according to whether or not reads are allowed to execute on one or both disks, the number of required queues, and the amount of concurrency provided between different updates, different reads, and an update and a read. We study the performance of these policies through the use of simple Markovian models and simulation. The analysis assumes exponential service times whereas the simulation accounts for real seek times, latencies, and transfer delays. We observe that policies which assign one of the disks to be a standby spare without using it to service read requests exhibit the worst performance. We also observe the best policy to be one that requires a central queue to hold all requests that have not initiated service and an auxiliary queue for storing write requests to the disk that lags behind. If the disks are distant from each other, then a simple policy that associates a local queue with each disk provides reasonable service. Last, we examine the effects of load balancing on performance and scheduling read requests to the same data item on the two disks so as to minimize read response times.

*This work is supported in part by the Office of Naval Research under contract N00014-87-K-796, and NSF equipment grant CERDCR 8500332

†The third author was in the Dept. of Electrical and Computer Engineering. He is presently with Digital Equipment Corporation.

1 Introduction

Many computer systems require highly reliable service. This requirement has spurred considerable interest in designing highly fault tolerant computer systems. These systems are typically designed to provide redundancy so as to survive single component failures. One example of such redundancy is the concept of *mirrored disk systems* where each data item is stored on a pair of disks [3, 8, 7]. Besides providing fault tolerance, disk mirroring also has the advantage of supporting multiple reads in parallel which has the potential of increasing the disk bandwidth and decreasing disk response times.

In this paper we will study the effects of different scheduling policies on the performance of a single mirrored disk system. These policies differ from one another according to whether both disks are allowed to service read requests, the number of queues required, and the level of concurrency provided between different requests. A combination of analysis and simulation is used to compare these different policies. Briefly, we observe

- The worst policies are those that do not allow reads to execute on both disks. This can occur in systems where one disk, a *secondary disk* is maintained in a *standby mode* until the other disk, commonly referred to as the *primary disk*, fails.

- The best policy is one that stores all requests in a single common queue prior to initiation of service and maintains an auxiliary queue with the disk that lags behind. This queue contains write requests to that disk that have already been serviced by the other disk. This policy is appropriate for an environment where the disks are in close proximity and a centralized controller implementation is feasible.

- If the disks are distant from each other and/or a centralized policy is infeasible, then a simple policy, whereby a fraction of the reads are routed to one disk and the remainder to the other disk, performs reasonably well. This policy maintains a *local queue* at each disk. The performance of such a policy can be improved by load balancing and the parallel submission of the same read request to each disk.

- The simultaneous submission of a read to both disks can improve performance, particularly at low loads. The read is considered to complete when the first operation on either disk completes.

The analysis used in this paper is also of interest. Many of the policies are analyzed using the matrix geometric methodology of Neuts [16]. It is our belief that this powerful and useful methodology deserves more attention than it has received by the performance evaluation community. Hence this paper illustrates well its application and its utility for handling complex system behavior.

Little work has appeared in the literature evaluating the performance of disk mirrored systems. Matloff [13] develops an approximate analysis of a different form of disk redundancy where *three or more disks* are used to maintain two copies of each data item. The performance of this system is shown to be better than that of a single disk that maintains a single copy of each data item. Bitton and Gray [4] examine a similar policy in the context of K disks. However, they do not account for queuing delays. Related work can be found in the area of replicated database systems. Nelson and Iyer [14] describe and analyze, using the matrix geometric methodology, two protocols for a replicated database that correspond to our CR-ESQ and CRU-ESQ policies. Other work on the performance evaluation of replicated data base systems can be found in [5, 6, 1].

As mentioned earlier, disk mirroring has the advantage of supporting simultaneous multiple reads. In addition, by allowing a read request to access the disk where the arm is currently

positioned closest to the target data, the access time can be reduced. Both of these actions have the effect of increasing I/O bandwidth. There exist other techniques such as *disk striping* [17] and *disk interleaving* [9] that also improve I/O bandwidth; however they do not support fault tolerance. An alternate way of improving I/O efficiency is to optimize the data layout on the disk so as to increase the locality of access. The disk locality issues will be treated later in the paper.

The remainder of this paper is organized as follows: Section 2 describes the various policies that we study. Section 3 describes the analysis of these policies. The performances of the policies are compared and discussed in Section 4 and the paper summarized in Section 5.

2 Description of Policies

In this section we describe the behavior of seven policies for scheduling read and update requests in a mirrored disk system. They can broadly be divided into two classes according to whether they require a single queue of waiting requests or multiple queues of waiting requests. As a general rule, multiple queue policies are only little more complicated to implement but provide much better performance.

2.1 Single Queue (SQ) Policies

In this category, every incoming request joins a common queue waiting for service. In all cases, requests are served in a first-come first-serve (FCFS) manner. Update requests are required to wait until both disks are idle. When an update begins its service, it initiates writes at each disk. Reads are handled differently by the different policies. This class of policies can be further subdivided into two subclasses depending on whether reads are allowed to execute on a single disk, the primary disk, or on both disks.

2.1.1 Primary/Secondary (PSSQ) Policies

One of the disks is termed the *primary* disk and the other the *secondary* disk.

- **Serial PSSQ (S-PSSQ) Policy.** This is the simplest policy which allows only one request to execute at any one time. No concurrency is allowed between either 1) two different reads, 2) two different updates, or 3) a read and an update. All requests at the head of the queue must wait until both disks are idle.

- **Concurrent PSSQ (C-PSSQ) Policy.** This policy differs from the previous one by allowing concurrency between a read and an update. Specifically a read may enter service anytime the primary disk is idle.

2.1.2 Equitable SQ (ESQ) Policies

Policies in this class treat both disks in an equitable fashion. The first two policies differ from each other in the amount of concurrency allowed between requests. The third policy differs from the other policies and the PSSQ policies by allowing a read request to execute on both disks. Under this policy, the read completes as soon as the first read operation completes on either disk.

- **Concurrent Read ESQ (CR-ESQ) Policy.** Read requests may execute on either disk. However, only one update is allowed to execute at any one time and no concurrency is allowed between a read and an update. Two reads may execute concurrently.

- **Concurrent Read Update ESQ (CRU-ESQ) Policy.** This policy improves on the previous policy by allowing concurrency between reads and updates. An update is still required to wait until both disks are available but a read request may proceed as soon as a disk becomes available.

- **Minimum Read ESQ (MR-ESQ) Policy.** This policy is a variation of the S-PSSQ policy. Each request is still required to wait until the previous request completes. Whenever a read begins service, it executes on both disks. The read completes as soon as the first read operation completes. The remaining read operation is aborted. This feature is similar to one studied in [13, 4] where the disk whose head is closest to the desired track is chosen.

2.2 Multiple Queue (MQ) Policies

One common feature of the single queue policies is that updates may proceed only when both disks are free. By introducing a second queue, the multiple queue policies allow updates to spawn write requests that can initiate asynchronously.

- **Distributed MQ (DMQ) Policy.** This policy maintains two separate queues, one for each disk. Every update generates two write requests, one for each queue. Reads randomly choose one of these queues to enter. In general, they may choose queue 1 with probability α_1, and queue 2 with probability $\alpha_2 = 1 - \alpha_1$. Once queued, requests are not allowed to switch queues. We will only consider the performance when $\alpha_1 = \alpha_2 = 1/2$. We refer to this as the Random DMQ (R-DMQ) policy.

  We also consider several variations of this policy. The first one termed the *shortest queue DMQ (SQ-DMQ)* policy requires incoming reads to join the shortest queue. The second and third variations allow a read to spawn two read operations that join each of the two queues. Under both variations, the read completes as soon as the first associated read operation completes. The variations differ from each other according to whether the second read operation is aborted or not. We refer to these two variations respectively as *minimum read DMQ (MR-DMQ)* and *minimum read DMQ with abort (MR-DMQA)* policies. The latter variation can be further subdivided into two policies according to whether the abort of the second read is performed at the time that the first read begins service, or as soon as it completes service

- **Common MQ (CMQ) Policy.** This policy maintains a common queue for all arriving requests. The request at the head of the queue may begin execution as soon as either disk is free. If the request is an update, it spawns two write requests, one of which begins service at the idle disk and the other which either initiates service on the second disk if it is idle, or queues up at that disk if it is busy. Hence, a queue of write requests may develop at the disk that lags behind under this policy. Requests in this queue are served in a FCFS manner.

Remark. The basic R-DMQ policy is implementable in any environment, regardless of the proximity of the two disks to each other. The CMQ policy is only suited to an environment where the disks are in close proximity and a central controller is possible. The variations on the R-DMQ policy are also better suited to a centralized environment.

For easy of reference, all these policies above are summarized in Table I and II.

3 Model and Analysis

In order to obtain the performance measures of the seven strategies described in the previous section, we assume that requests arrive according to a Poisson process with rate λ, and that the service times at each disk form two mutually independent exponentially distributed r.v.'s with average $1/\mu$. The independence assumption between service times at each disk is reasonable since the copies of the same data item may be stored in different cylinders, tracks, or blocks on the two disks. Let p_r be the probability that an arriving request is a read and $p_w = 1 - p_r$ be the probability that a request is an update. We further assume that disk failures occur infrequently so as to have little effect on the performance of the different policies.

Under these assumptions, the S-PSSQ and MR-SQ policies can be modeled as $M/G/1$ queues and the other policies can be modeled by multi-dimensional Markov chains. The basic technique that we use to solve these Markov chains is the *matrix geometric method* developed by Neuts [16]. The C-PSSQ, CR-PSSQ, and CRU-PSSQ policies can be solved exactly using this method whereas the MQ policies are solved approximately. In the latter case the approximate analysis yields upper and lower bounds for the performance measures. The performance measures we are mainly interested in are the mean read and update response times, as well as the overall mean response time and the maximum throughput.

3.1 The S-PSSQ and MR-ESQ Policies

All I/O requests are executed serially under both policies. Neither policy provides any concurrency between different requests. Consequently, they can be each modeled as an $M/G/1$ queue.

S-PSSQ Policy: The service time for a read, $X^{(r)}$, is a exponential r.v. with probability density function (p.d.f.) $f_{X^{(r)}}(x) = \mu e^{-\mu x}$. Let X_1 and X_2 denote service times at each disk. The update service time $X^{(u)}$ is the maximum of these two times, $X^{(u)} = \max\{X_1, X_2\}$. Since X_1 and X_2 are independent exponential r.v.'s with mean $1/\mu$, the p.d.f. for $X^{(u)}$ is $f_{X^{(u)}}(x) = 2\mu e^{-\mu x}(1 - e^{-\mu x})$. Therefore. the p.d.f. for the overall service time, X, is

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

with first and second moment

$$\begin{aligned} E[X] &= \frac{3 - p_r}{2\mu}, \\ E[X^2] &= \frac{7 - 3p_r}{2\mu^2}. \end{aligned}$$

The average waiting time, given by the Pollaczek-Khinchine formula [11, p. 187], is

$$W_q = \frac{\rho \overline{x}(1 + c_b^2)}{2(1 - p_r)} \tag{1}$$

where $\rho = \lambda E[X]$ and c_b is the coefficient of variation of the service time, $c_b^2 = E[X]^2 / E[X^2]$. The average response times for reads and updates are

$$W_r = W_q + \frac{1}{\mu} \tag{2}$$

$$W_u = W_q + \frac{3}{2\mu}. \tag{3}$$

Table I: Single Queue Policies (SQ):

Primary/Secondary Model (PSSQ)	*Equitable Model* (ESQ)
• **S-PSSQ:** All requests at the head of queue must wait until both disks become idle. • **C-PSSQ:** A read may enter service anytime the primary disk is idle. An update must wait until both disks are free.	• **CR-ESQ:** Two reads may execute concurrently on the two disks. An update have to wait until both disks are free. No *read-update* concurrency is allowed. • **CRU-ESQ:** Improve CR-ESQ by allowing *read-update* concurrency. An update is still need to wait until both disks are available. • **MR-ESQ:** Similar to S-PSSQ, but a read executes on both disks, and finishes as soon as the first read completes. The other is aborted.

Table II: Multiple Queue Policies (MQ):

Distributed Model (DMQ)	*Centralized Model* (CMQ)
• **R-DMQ:** Two separate queues are maintained, one for each disk. Reads randomly join one queue. An update generates two write requests, one for each queue. • **SQ-DMQ:** Updates are as above. Reads join the shortest queue. • **MR-DMQ:** A read also spawn two reads that join each of these two queues. It completes as soon as the first read terminates. Updates are the same as above. • **MR-DMQA:** The same as MR-DMQ, except that when the first read finishes, the other is aborted.	• **CMQ:** maintains a common queue for all arriving requests. An update can also proceed as soon as either disk is free. If the other disk is busy, it spawns a write operation to join an auxiliary queue built up at the disk lags behind.

MR-ESQ Policy: The write service time is the same as in policy 1, but the read service time is now the minimum of two disk service times, $X^{(r)} = \min(X_1, X_2)$ and its p.d.f. is $f_{X^{(r)}}(x) = 2\mu e^{-2\mu x}$. The first two moments of the service time are

$$E[X] = \frac{3 - 2p_r}{2\mu},$$
$$E[X^2] = \frac{7 - 6p_r}{2\mu^2}.$$

Formulas (1) and (3) can be used to obtain the mean update response time. Equation (1) along with

$$W_r = W_q + 1/(2\mu)$$

yield the mean read response time.

Remark. The M/G/1 model for this and the S-PSSQ policy can be easily extended to account for more realistic disk service times that include seek, latency, and transfer time.

3.2 The C-PSSQ, CR-ESQ, and CRU-ESQ Policies

The performance measures for each of these policies can be obtained by studying the behavior of a multidimensional Markov chain. The analyses differ according to the state definitions and the infinitesimal generators associated with the Markov chains. As before we assume a Poisson arrival process and exponential disk service time. We will describe the performance analysis of the C-PSSQ policy in some detail. In the case of the other two strategies, we will only give their state descriptions. The infinitesimal generators associated with these processes are not difficult to express given the state definitions and the description of the behavior of the strategies found in the preceding section.

C-PSSQ Policy: We define a two-dimensional stochastic process $[N(t), S(t)]$, where $N(t)$ denotes the number of requests in the waiting queue and $S(t)$ denotes the status of the two disks. Since we distinguish the primary and secondary disks, $S(t)$ may take one of the following values:

$$S(t) = \begin{cases} 0 & \text{if both disks are idle,} \\ R & \text{if only primary disk is reading,} \\ WR & \text{primary disk is reading and secondary is writing,} \\ W_p & \text{if only the primary disk is writing,} \\ W_s & \text{if only the secondary disk is writing,} \\ 2W & \text{if both disks are writing.} \end{cases}$$

Based on our assumptions of Poisson arrivals and exponential service time, $[N(t), S(t)]$ is a Markov chain. We are only interested in the stationary behavior of the system, when it exists. Consequently we define $N = \lim_{t\to\infty} N(t)$, $S = \lim_{t\to\infty} S(t)$, and $p(i,j) = \Pr(N = i, S = j)$, $i = 0, 1, \cdots$, $j \in \{0, R, WR, W_p, W_s, 2W\}$.

Let

$$y(0) = [p(0,0), p(0,R), p(0,W_p), p(0,W_s), p(0,2W)]$$

and

$$y(i) = [p(i,R), p(i-1,WR), p(i,W_p), p(i,W_s), p(i,2W)] \quad i = 1, 2, \cdots.$$

Then $Y = (y(0), y(1), ..., y(i), ...)$ is the stationary probability vector for the process $[N(t), S(t)]$. The infinitesimal generator Q,satisfying $YQ = 0$, is

$$Q = \begin{pmatrix} B_1 & B_0 & 0 & 0 & 0 & . \\ B_2 & A_1 & A_0 & 0 & 0 & . \\ 0 & A_2 & A_1 & A_0 & 0 & . \\ 0 & 0 & A_2 & A_1 & A_0 & . \\ . & . & . & . & . & . \end{pmatrix}$$

where the matrices B_0, B_1, B_2, A_0, A_1, and A_2 are given by,

$$B_0 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ \lambda & 0 & 0 & 0 & 0 \\ 0 & 0 & \lambda & 0 & 0 \\ 0 & p_r\lambda & 0 & p_w\lambda & 0 \\ 0 & 0 & 0 & 0 & \lambda \end{pmatrix}$$

$$B_1 = \begin{pmatrix} -\lambda & p_r\lambda & 0 & 0 & p_w\lambda \\ \mu & -(\lambda+\mu) & 0 & 0 & 0 \\ \mu & 0 & -(\lambda+\mu) & 0 & 0 \\ \mu & 0 & 0 & -(\lambda+\mu) & 0 \\ 0 & 0 & \mu & \mu & -(2\mu+\lambda) \end{pmatrix}$$

$$B_2 = \begin{pmatrix} 0 & p_r\mu & 0 & 0 & p_w\mu \\ 0 & \mu & 0 & \mu & 0 \\ 0 & p_r\mu & 0 & 0 & p_w\mu \\ 0 & 0 & 0 & 0 & \mu \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$A_0 = \lambda \cdot I_{5\times 5}$$

$$A_1 = \begin{pmatrix} -(\mu+\lambda) & 0 & 0 & 0 & 0 \\ 0 & -(2\mu+\lambda) & 0 & 0 & 0 \\ 0 & 0 & -(\mu+\lambda) & 0 & 0 \\ 0 & 0 & 0 & -(\mu+\lambda) & 0 \\ 0 & p_r\mu & \mu & p_w\mu & -(2\mu+\lambda) \end{pmatrix}$$

$$A_2 = \begin{pmatrix} p_r\mu & 0 & 0 & 0 & p_w\mu \\ \mu & p_r\mu & 0 & p_w\mu & 0 \\ p_r\mu & 0 & 0 & 0 & p_w\mu \\ 0 & 0 & 0 & 0 & \mu \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Here $I_{n\times n}$ denotes the $n \times n$ identity matrix. Observe that the infinitesimal generator has a tridiagonal form with submatrices as its elements. Neuts [16] investigated Markov chains having infinitesimal generators of this form and determined their stability conditions to be

$$\pi A_2 e > \pi A_0 e$$

where π is the unique vector satisfying $\pi > 0, \pi A = 0$, and $\pi e = 1$. The matrix A is defined as $A = A_0 + A_1 + A_2$, and e is a column vector containing all 1's.

Neuts showed that the elements of stationary probability vector Y can be represented in a matrix geometric form,

$$y(i) = y(1)R^{i-1}$$

where R is the minimal solution of $A_0 + RA_1 + R^2A_2 = 0$. The matrix R can be obtained iteratively using the following equations,

$$\begin{aligned} R(0) &= 0, \\ R(n+1) &= -A_0A_1^{-1} - R^2(n)A_2A_1^{-1}, \quad n \geq 0. \end{aligned}$$

The stationary probability vector Y is obtained by solving the equations,

$$\begin{aligned} (y(0), y(1)) \begin{pmatrix} B_1 & B_0 \\ B_2 & A_1 + RA_2 \end{pmatrix} &= 0, \\ y(0)e + y(1)[I - R]^{-1}e &= 1. \end{aligned} \tag{4}$$

The stability condition for this policy is $\lambda < \mu(2 - p_r)/(3 - 3p_r + p_r^2)$.

Once the stationary probability Y is known, the desired performance measures can be easily computed. The average queue length, $E[N]$, is

$$\begin{aligned} E[N] &= \sum_{i=0}^{\infty}[ip(i,R) + ip(i,WR) + ip(i,W_p) + ip(i,W_s) + ip(i,2W)] \\ &= \sum_{i=0}^{\infty} iy(i)e - \sum_{i=0}^{\infty} y(i)e_2 \\ &= y(1)[I - R]^{-2}e - y(1)[I - R]^{-1}e_2 \end{aligned}$$

where e_j is the column vector consisting of all 0's except the jth position which contains a 1.

By Little's result, the average waiting time in the queue is

$$W_q = \frac{E[N]}{\lambda}$$

therefore

$$\begin{aligned} W_r &= W_q + \frac{1}{\mu}, \\ W_u &= W_q + \frac{3}{2\mu}. \end{aligned}$$

CR-ESQ Policy: This policy treats two disks equally. The state description has the same form as for the preceding policy with $N(t)$ unchanged and $S(t)$ is defined as,

$$S(t) = \begin{cases} 0 & \text{if both disks are idle,} \\ R & \text{if only one disk is reading,} \\ 2R & \text{if both disks are reading,} \\ W & \text{if only one disk is writing,} \\ 2W & \text{if both disks are writing.} \end{cases}$$

The stationary probability vector is $Y = (y(0), y(1), \cdots)$ where

$$\begin{aligned} y(0) &= [p(0,0), p(0,R), p(0,W), p(0,2W)], \\ y(i) &= [p(i-1,2R), p(i,R), p(i,W), p(i,2W)], \quad i = 1, 2, \cdots. \end{aligned}$$

and the stability condition is given by

$$\lambda < \frac{2\mu}{3 - p_r - p_r^2}.$$

Expressions for the expected response time of an update and a read can be obtained using arguments similar to those used for the preceding policy [14].

CRU-ESQ Policy: In this policy $N(t)$ is the same as before, and $S(t)$ simply denotes the number of busy disks at time t, $S(t) = 0, 1, 2$.

The elements of Y are defined by,

$$\begin{aligned} y(0) &= [p(0,0), p(0,1)], \\ y(i) &= [p(i-1,2), p(i,1)], \quad i = 1, 2, \cdots \end{aligned}$$

and the stability condition in this case is

$$\lambda < \frac{2\mu}{3 - 2p_r}.$$

3.3 The MQ Policies

In these two cases, the behavior of each system is represented by a Markov chain with at least two state variables that can take on an infinite number of different values. Consequently, the matrix geometric method is no longer directly applicable. In both cases, we use results from [18] where the system behavior is approximated by a Markov chain with only one state variable that can take an infinite number of values. We briefly describe the basic ideas behind the analysis of each policy.

R-DMQ Policy: This policy maintains a separate queue at each disk. If the arrival process is Poisson and the read requests are routed to the two disks according to a Bernoulli process, then each queue behaves as a $M/M/1$ queue with Poisson arrival rate $\alpha_i p_r \lambda + p_w \lambda$, where p_r, p_w and $\alpha_i, i = 1, 2$ are defined as before. Consequently, the performance metrics associated with read requests can be easily obtained from an M/M/1 model. However, the behavior of the updates is more complicated and requires a more detailed study of the two queues.

This queuing system is an example of a class of systems referred to as *fork-join* queuing systems. Numerous papers have studied the behavior of these systems, [2, 10, 15, 18]. We briefly describe the approach taken in the last of these papers as it is the one that we use to obtain performance metrics of the updates.

The system behavior can be represented by a two-dimensional Markov chain, $[N_1(t), N_2(t)]$ where $N_i(t)$ denotes the number of requests present at time t at the i-th disk, $i = 1, 2$. As mentioned above, this system is approximated by a system where one of the queue lengths, say $N_2(t)$, is never allowed to exceed some integer $B > 0$. This modified system behaves in the following manner,

- whenever a read request arrives to disk 2 and finds B requests present, it is rejected.
- whenever an update arrives and finds B requests at the second disk, a write request is generated only for the first disk.

Based on this idea, [18] develops a matrix geometric model of the modified system which is used to provide upper and lower bounds on the performance of updates. The model allows one to tradeoff accuracy in the bounds with respect to computational requirements. In this paper, we report results that are always within 2% of the correct result. The reader is referred to [18] for details of the analysis.

CMQ Policy: This policy maintains a common queue for all arrivals. In addition a queue is maintained for each disk which contains requests for that particular disk. At any point in time, only one of these auxiliary queues will contain requests waiting for service. These requests waiting for service will always be write requests for which the corresponding request on the other disk is either in service or has completed.

The behavior of the system is represented by a three dimensional Markov chain $[N_1(t), N_2(t), N_3(t)]$ where arrivals are characterized by a Poisson process and service times of all requests on either disk form a sequence of i.i.d. exponential r.v.'s. Here $N_1(t)$ is the number of requests in the common queue, $N_2(t)$ is the number of write requests in the auxiliary queue which contains waiting requests, and $N_3(t)$ indicates whether the other disk is busy or idle, $N_3(t) = 0, 1$. Since both $N_1(t)$ and $N_2(t)$ can take any nonnegative integer value, it is not possible to apply the Matrix Geometric methodology to this problem. It is possible to develop lower and upper bounds on the performance metrics of interest by suitably truncating one of these variables and properly modifying the resulting Markov chain. This has been performed in [18]. The numerical results that we will present in the next section are obtained using the results from that analysis.

Remark. The same analysis applies to the DMQA policy where the abort of the second read occurs as soon as the first read begins service.

4 Performance Comparisons

In this section, we compare the performance of the different policies. For the numerical results, we assume the mean service rate $\mu = 45$ (requests/sec), in order to be consistent with the parameters in our real disk simulations. We have compared the throughput and mean response time of updates and reads for the seven policies. Figure 1 shows the maximum throughput of the seven policies as a function of the read probability p_r. All policies exhibit throughputs that are increasing functions of p_r. As expected, the MQ policies provide the best throughputs. The equitable policies fall in a middle group performing as well as the MQ policies when most requests are reads but as poorly as the PSSQ policies when most requests are updates. Interestingly enough, the variation on the S-PSSQ policy whereby a read spawns separate read requests at each disk (the MR-ESQ policy) provides a significant improvement in throughput as p_r approaches one. Matloff has made similar observations in his study [13].

The mean response times of reads and updates under the SQ policies are given in figure 2 as a function of the arrival rate of all requests when $p_r = 0.75$. As the arrival rate increases, the PSSQ policies yield poor performance compared to the equitable SQ policies. It is again interesting to observe that the MR-ESQ policy yields a significant improvement in the mean response time of updates and reads. The mean read response time is lower under the MR-ESQ policy than under the other policies, particularly for low arrival rates. This is due to the fact that a read completes as soon as the first of the two associated read operations completes. We will observe that a similar enhancement to R-DMQ policy also lowers the mean read response time under low loads.

The mean response times under the best two SQ policies and the two MQ policies are compared in figure 3 as a function of the request arrival rate when $p_r = 0.75$. We observe a significant difference between the SQ and MQ policies over most arrival rates. The MQ policies improve

performance because of the removal of the requirement that updates not begin service until both disks are idle. The exception to this observation occurs at very low arrival rates where the MR-ESQ policy yields lower mean read response times for reasons given earlier in this section. In addition, we observe that the CMQ policy yields lower mean response times than the R-DMQ policy. This is particularly apparent in the case of the mean update response time.

In the remainder of this section, we report on additional results obtained for the MQ policies via simulation. These studies assume more realistic service times. Specifically, the disk service time is

$$X = U \cdot X_S + X_L + X_T$$

where X_S, X_L, and X_T are *r.v.*'s denoting the *seek, latency* and *transfer* times, respectively and U is a binary valued *r.v.* which indicates the presence or absence of a seek, $U = 1$ whenever a seek is required and $U = 0$ otherwise. The seek and latency times are assumed to be uniformly distributed r.v.'s. The transfer time is assumed to be constant, corresponding to the transfer of a fixed length block of data. Taking the IBM's 3330-11 disk driver as an example [12], the average seek time is 30 msec, average latency time 8 msec, and the transfer time 2 msec. We define a parameter, *the probability of sequential access*, p_s, to be the probability that a request is made to the same cylinder as that of the previous request, i.e., $\Pr[U = 0] = p_s$ and $\Pr[U = 1] = 1 - p_s$. We assume $p_s = 0.6$ in most of simulations except when indicated otherwise.

Our simulations use these parameters to simulate disk service times. The regenerative method is employed to obtain 95% *confidence intervals* in all of these simulations. Here the regeneration points are defined as the times when the system empties, i.e., both disks become idle. A regeneration cycle between two contiguous regeneration points includes an idle period followed by a busy period (a busy period begins when a request arrives to an empty system). The performance measures are obtained such that the length of confidence interval is less than 2% of the estimated value of the response time. Figure 4 shows the numerical and simulation results for the CMQ policy. Although the two curves do not quite match because of the exponential service time assumption made in our analysis, their qualitative behavior remains the same.

In figure 5 we compare the predictions made by the model to those obtained by simulation as a function of arrival rate for different probabilities of sequential access. As expected, with the same workload, the mean response time is a decreasing function of the p_s. As workload increases, the differences between the mean response time for each p_s increase dramatically. Also we observe a better match between the analytic and simulation results as the sequential access probability p_s increases, especially for the mean update response time.

We complete this section with a discussion of the performance of the different variations on the DMQ policy. Figure 6 illustrates the performance of the different variations as a function of workload. As expected, the response time is an increasing function of workload. We observe that the SQ-DMQ, MR-DMQA, and CMQ policies perform better than the R-DMQ policy, and that the CMQ policy is the best except at low utilizations where the MR-DMQA policy yields lower response times for the reads.

5 Summary

In this paper we have described and evaluated the performance of numerous policies for scheduling I/O requests to a mirrored disk system. We observed that the best performance is given by a centralized policy which allows reads to execute on either disk and concurrent execution by any two types of I/O requests. The worst performance is given by a policy that treats one of the disks

as a redundant standby which cannot be used to service reads. The performances of the remaining policies fall in between and are ordered primarily by the capabilities of the policies to execute reads on both disks and by the amount of concurrency provided between different I/O requests. Last, we studied the effect of initiating two read operations, corresponding to a single read request, on different disks. If the read is considered to complete as soon as the first operation completes, then some decrease in mean read response time is possible under low loads.

References

[1] F. Baccelli and E.G. Coffman, Jr., "A data base replication analysis using an M/M/m queue with service interruptions", *Performance Evaluation Review* Vol.11 (4), pp.102-107, 1982-1983.

[2] F. Baccelli, A. Makowski, "Simple computable bounds for the fork-join queue." *Proc. Conf. Inform. Sci. Systems*, 1985.

[3] J.F. Bartlett, "A Nonstop Kernel", *Proc. Eighth Symp. on Operating System Principles*, pp.22-29, 1981.

[4] D. Bitton, J. Gray, "Disk Shadowing", *Proc. 14-th VLDB Conf.*, Los Angeles, CA, Aug. 1988.

[5] E.G. Coffman, Jr., H.O. Pollak, E.Gelenbe, and R.C. Wood, "An Analysis of Parallel-read Sequential-write Systems", *Performance Evaluation* Vol. 1, pp.62-69, 1981.

[6] E.G. Coffman, Jr., E.Gelenbe and B. Plateau, "Optimization of the number of copies in a distributed data base", *IEEE Trans. on Software Engi.*, 7(1) pp.78-84, 1981.

[7] C.P. Grossman, "Evolution of the DASD Control", *IBM Systems Journal*, Vol. 28, No. 2, 1989.

[8] J. Katzman, "A Fault-tolerant Computing System", in *The Theory and Practice of Reliable System Design*, D. Siewiorek and R. Swarz Eds., Digital Press, 1982.

[9] M.Y. Kim, "Synchronized Disk Interleaving," *IEEE Trans. on Computers*, Nov. 1986.

[10] C. Kim and A.K. Agrawala. "Analysis of a Fork-Join Queue." *IEEE Trans. on Comp.*, Vol. 38 (2), pp. 250-55, Feb. 1989.

[11] L. Kleinrock, *Queueing Systems, Vol. 1*, John Wiley, New York, 1975.

[12] S.S. Lavenberg, *Computer Performance Modeling Handbook* Academic Press, Inc. 1983.

[13] N.S. Matloff, "A Multiple-Disk System for Both Fault Tolerance and Improved Performance", *IEEE Trans. on Reliability*, Vol. 36, 2, pp. 199-201, June 1987.

[14] R.D. Nelson and B.R. Iyer, "Analysis of a Replicated Data Base", *Performance Evaluation* Vol. 5, pp.133-148. 1985.

[15] R. Nelson and A.N. Tantawi. "Approximate analysis of fork/join synchronization in parallel queues." *IEEE Trans. Computers*, Vol. 37, pp. 739-743, 1988.

[16] M.F. Neuts, *Matrix Geometric Solutions in Stochastic Models - an Algorithmic Approach*, John Hopkins Univ. Press., 1981.

[17] K. Salem and H. Garcia-Molina, "Disk Striping," *Proceedings 1986 Data Engineering Conf.*, Los Angeles, Feb. 1986.

[18] D. Towsley and S.-P. Yu, "Bounds for Two Sever Fork-Join Queueing Systems", COINS Tech. Rep. 87-123, U. Massachusetts, Nov. 1987.

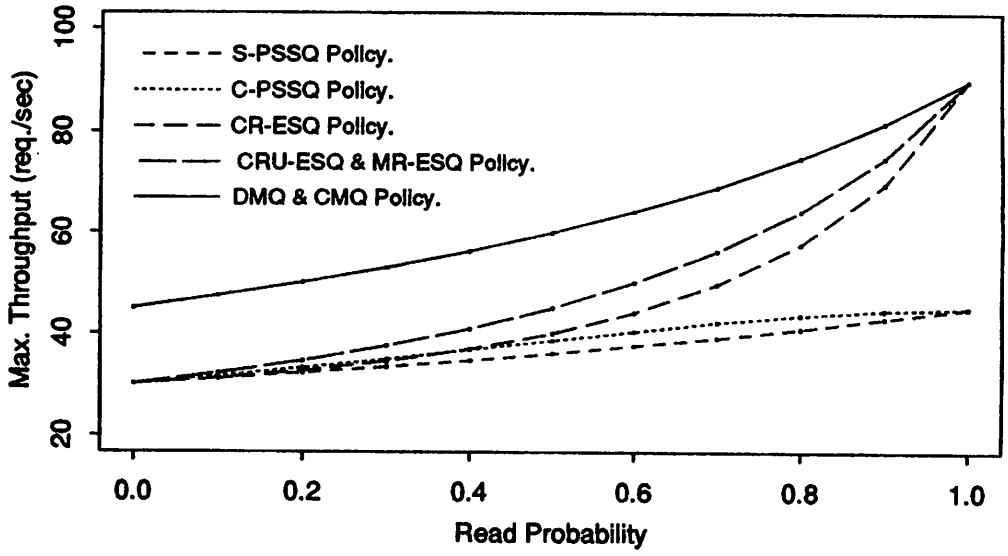

Figure 1: Maximum Throughput of the Different Policies. ($\mu = 45$)

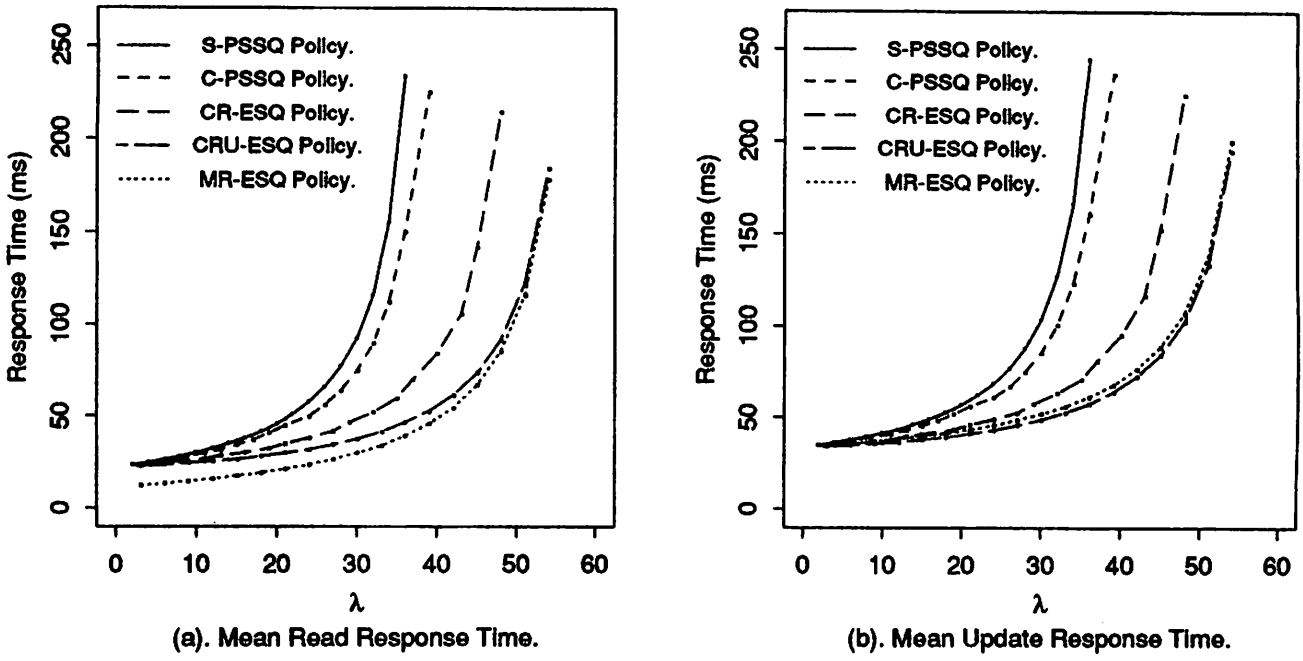

Figure 2: Mean Response Time for the SQ Policies. ($p_r = 0.75, \mu = 45$)

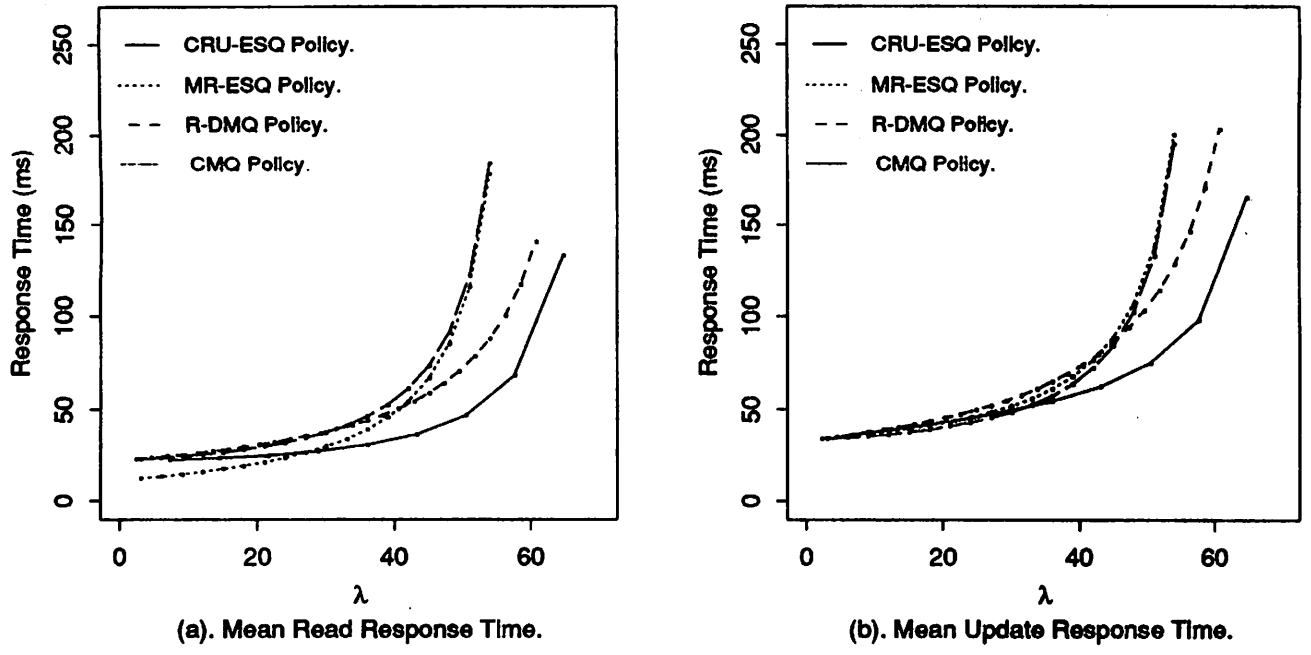

Figure 3: Comparison of the SQ and MQ Policies. ($p_r = 0.75, \mu = 45$)

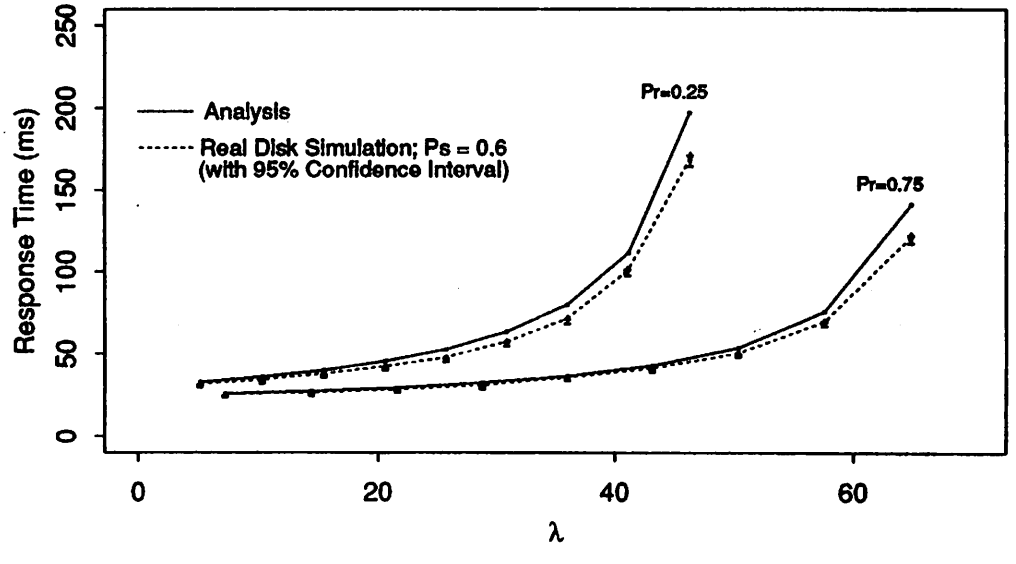

Figure 4: Mean Response Time for the CMQ Policy. ($p_r = 0.25,\ 0.75, \mu = 45$)

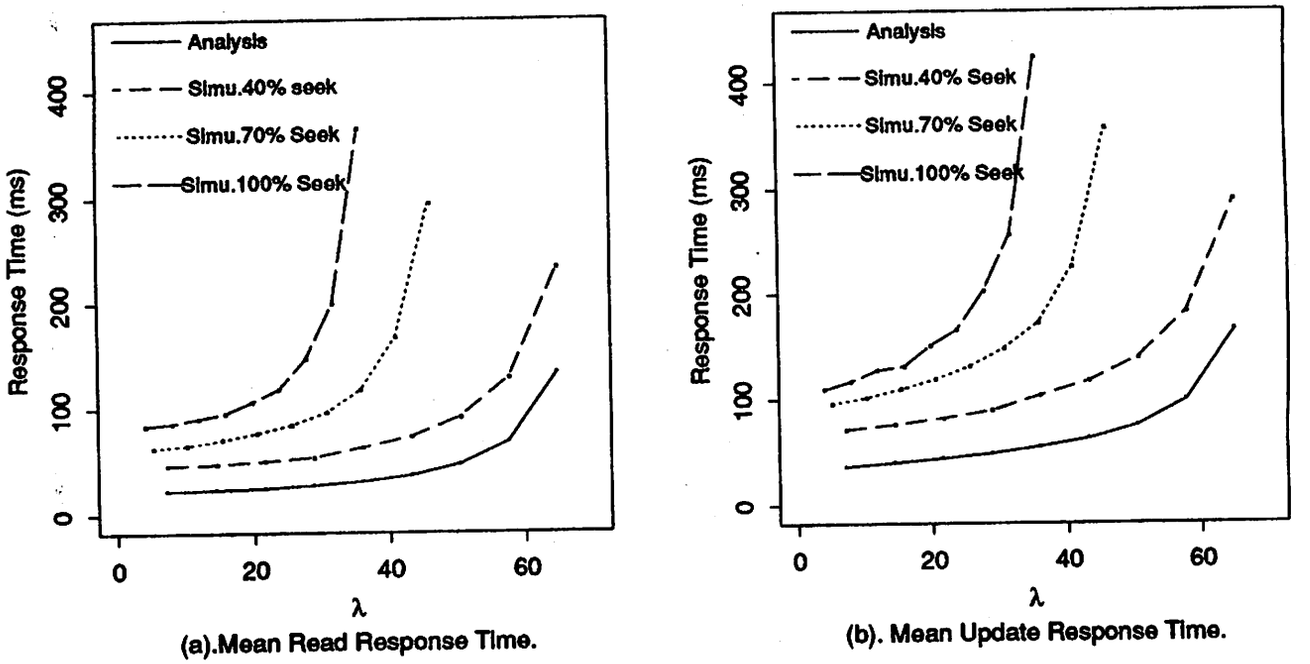

Figure 5: CMQ Policy with Different Sequential Access Probabilities. ($p_r = 0.75, \mu = 45$)

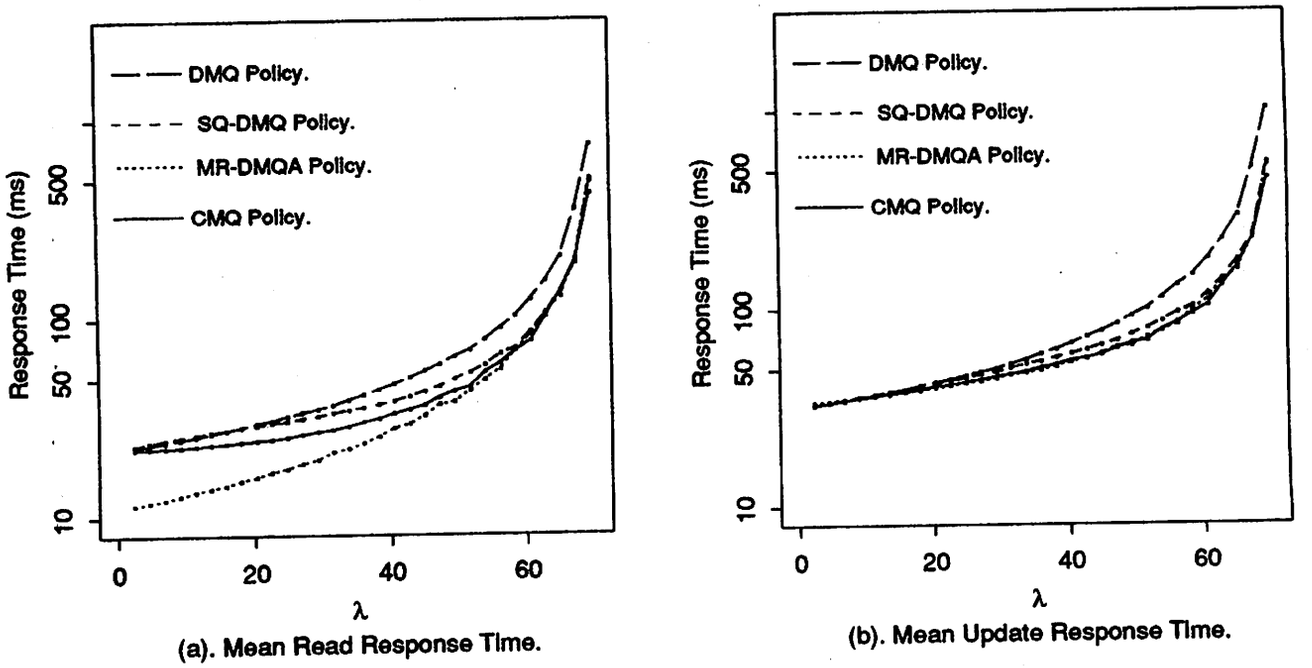

Figure 6: Simulations of the variations in the DMQ policyand the CMQ. ($p_r = 0.75, \mu = 45$)