Performance Analysis of Disk Arrays Under Failure*

Richard R. Muntz and John C.S. Lui
UCLA Computer Science Department, LA, CA 90024-1596, USA

Abstract

Disk arrays (RAID) have been proposed as a possible approach to solving the emerging I/O bottleneck problem. The performance of a RAID system when all disks are operational and the $MTTF_{sys}$ (mean time to system failure) have been well studied. However, the performance of disk arrays in the presence of failed disks has not received much attention. The same techniques that provide the storage efficient redundancy of a RAID system can also result in a significant performance hit when a single disk fails. This is of importance since single disk failures are expected to be relatively frequent in a system with a large number of disks. In this paper we propose a new variation of the RAID organization that has significant advantages in both reducing the magnitude of the performance degradation when there is a single failure and can also reduce the $MTTF_{sys}$. We also discuss several strategies that can be implemented to speed the rebuild of the failed disk and thus increase the $MTTF_{sys}$. The efficacy of these strategies is shown to require the improved properties of the new RAID organization. An analysis is carried out to quantify the tradeoffs.

1 Introduction.

A disk array is a set of disks with redundancy to protect against data loss. Patterson [7] describes several methods of organizing the data on disks including mirroring and multiple data blocks plus parity. Compared with mirroring, the "$N + 1$" RAID organization sacrifices some performance in terms of I/O access rate but with a reduced storage penalty for redundancy. In the remainder of this paper we will use the term RAID to refer to the $N + 1$ RAID organization. We also concentrate exclusively on the "small read/small write" use of the disks in which they are accessed independently rather than the mode in which large reads (writes) are performed by concurrent reads (writes) executed on each disk.

Two aspects of RAID systems have been examined most closely: performance under the condition that all disks are operational (e.g., [1]) and the mean time to system failure (data loss) (e.g. [4]). The performance of RAID when there are inoperable (or inaccessible) disks has largely been ignored (an exception is [8]).

Compared to system failure, it is a relatively common occurrence to have a single disk unavailable and therefore the performance of the system during the repair period is of concern. The $N + 1$ RAID organization achieves a low cost in redundant storage overhead but at the price of requiring multiple reads to the surviving disks in the same array each time a block on the failed disk must be reconstructed (i.e., to satisfy a read request for that block). In the worst case (a workload of all reads and no writes) this can double the access rate to the surviving disks and thus in effect, cut the capacity of the array in half. Consider for example a shared nothing DBM architecture as in Figure 1. Each node is this system would have one or more disk arrays. The impact that this can have on total system performance is dependent on the characteristics of the system workload. We are particularly interested in database applications of the RAID architecture. The impact of a single disk failure is most severe in the case of a "decision support" environment in which complex queries are common and the database tables have been partitioned among the disks on all or many nodes for increased I/O bandwidth. Complex operations can be limited by any imbalance in the system, which can be caused

*This research was supported by IBM through a University of California MICRO grant.

Proceedings of the 16th VLDB Conference
Brisbane, Australia 1990

by skew in the load [6] or by a disk array with diminished capacity due to a failed disk [5]. In a one hundred disk system, a single failed disk will represent a loss of only 1% of the raw I/O capacity of the system. However, if the effect is to reduce the capacity of the array to which it belongs by say 25%, this can cause a significant imbalance in the system, and the impact on aggregate system performance can be considerable.

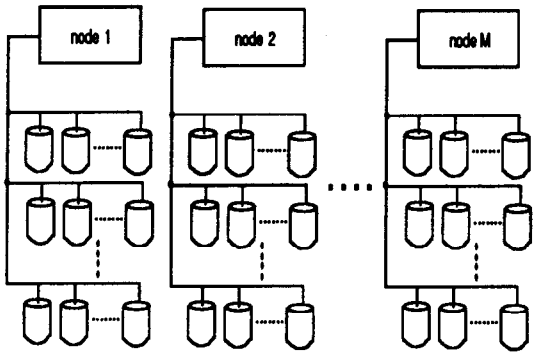

Figure 1: Shared Nothing DBM with disk arrays.

In order to maintain a reasonable $MTTF_{sys}$ (mean time to failure of the system) in a system with a large number of disks it is necessary to provide immediate repair of a failed disk. To accomplish this, "hot standby" disks are provided and the system automatically rebuilds the contents of the failed disk on the standby disk from the redundant information on the surviving disks [4]. The $MTTF_{sys}$ of the RAID is easily shown to be inversely proportional to the rebuild time [1, 3]. In the $N+1$ RAID system described in [7], to rebuild the failed disk contents at maximum speed (the capacity of the standby disk) would use the entire capacity of the surviving disks in the array. Thus to rebuild at maximum rate would mean that the array can perform no other work during the rebuild period. One can of course, tradeoff the rebuild rate with the rate at which the surviving disks process normal workload requests. However, this increases the time to rebuild the failed disk contents and thereby decrease the $MTTF_{sys}$. Quantifying this tradeoff is one of the purposes of this paper.

We also propose a new RAID organization. In the proposed architecture the same type of tradeoffs exist between normal workload and $MTTF_{sys}$ but the proposed organization can (for some system and workload parameters) yield an improved $MTTF_{sys}$ and support a higher workload during the rebuild period. This is accomplished by relaxing an assumption that has been made in most of the RAID work, i.e., that the "group size" (the number of data blocks plus parity) is the same as the "cluster" size (the number of disks over which the groups of blocks are distributed). We propose consideration of larger cluster sizes and show that by spreading the groups over a larger cluster of disks the increased load (per disk) on surviving disks can be significantly decreased. The cluster size will effect the time required to rebuild a failed disk (and therefore the $MTTF_{sys}$) and also the workload (measured in accesses per second per disk) that can be supported during the rebuild. These issues are addressed analytically in a later section. The results indicate that substantially better performance and reliability can often be obtained from such an organization with properly chosen values for group and cluster size.

The only study that we are aware of that considers disk subsystem performance under failure and the reconstruction of the failed disk is the Copeland and Keller study comparing mirroring and "interleaved declustering" [3]. The analysis of the proposed RAID organization presented here is similar with the methodology introduced in [3].

In section 2 we present the proposed new RAID organization and also several strategies for efficient rebuilding of a failed disk. In section 3 we present an outline of the analysis of the proposed techniques. Section 4 presents numerical results for a range of parameter values and their interpretation. Section 5 summarizes the contributions of the paper.

2 RAID Recovery.

It is assumed here (as in [3]) that there is a system requirement to provide some minimum level of service during the rebuild period which will be measured in terms of accesses per second per disk, denoted by λ_0. λ_0 is the access rate per "logical disk" and when an array contains a failed disk, the load on the surviving disks will actually be higher since the λ_0 accesses per second originally directed to the failed disk will require additional I/Os to the surviving disks. A consequence of the requirement for a minimum workload throughput is an additional limitation on the rate at which the failed disk can be rebuilt, i.e. the full capacity of the surviving disks is not available for rebuilding the contents of the failed disk. The cost of reconstructing the contents of a block from the failed disk causes the tradeoff between λ_0 and the rebuild rate to be of greater importance because the rebuild time becomes more sensitive to λ_0. Next we introduce

the *clustered RAID* architecture which separates the issue of group size and the cluster of disks over which the groups are distributed and then in the remainder of this section we define several disk rebuild strategies. In the following section we will use an analytic model to show that the proposed new RAID organization, in conjunction with the suggested rebuild strategies, can significantly improve the level of performance in the face of failures.

2.1 Clustered RAID.

We use the term "group" to refer to a set of N data blocks and 1 parity block that the $N+1$ RAID organization uses. As depicted in Figure 2, in the original $N+1$ RAID architecture the number of disks over which the groups are distributed is also equal to the group size. Define a *cluster* to be the set of disks over which the groups are distributed. The cluster size will be denoted by C and the group size by G where $G \leq C$. In the classic $N+1$ RAID architecture $N+1 = G = C$. Figure 3 depicts a cluster of size 5 with a group size of 4. (These small numbers are chosen for ease of presentation.)

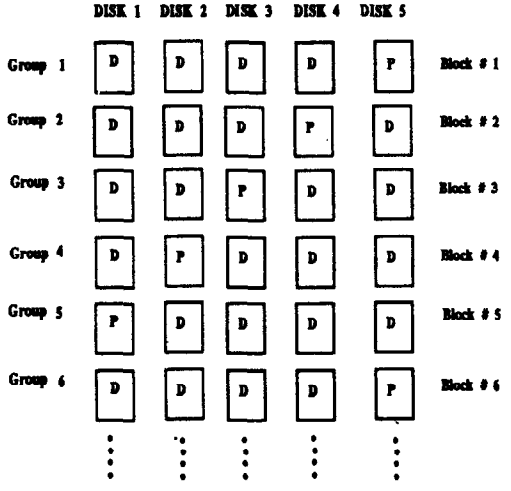

Figure 2: Data placement in $N+1$ RAID.

When a disk has failed, $G-1$ blocks must be read to reconstruct the contents of a block on the failed disk. However, by properly distributing the groups over the C disks in a cluster the extra load can be distributed evenly over the $C-1$ surviving disks in the cluster. In Figure 3 there are $\binom{5}{4}$ different ways to select the disks to hold a group. For each such combination there are 4 choices for which is the parity block. It is

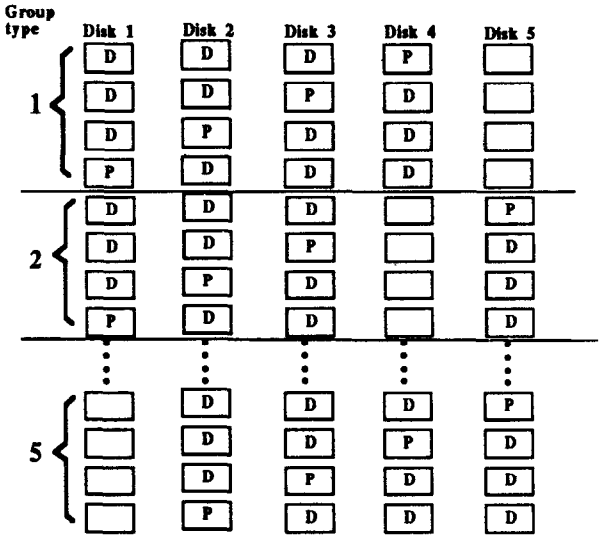

Figure 3: Each group type corresponds to one way to select 4 disks out of the five. Within a group type there are choices as to which disk holds the parity block.

not difficult to see that if groups are assigned in this manner then when any disk fails the extra load will be evenly distributed over the surviving disks. Note that the unlabeled blocks in the figure do *not* represent unused blocks (e.g. the data blocks for group type 2 on disk 5 are allocated to the first 4 physical blocks of disk 5).

This scheme does not require additional disks for a system (the storage overhead is determined by the group size G) and yet derives advantages from clustering the disks in sets larger than G. In this paper we will evaluate the benefits of this organization with respect to the performance under failure and $MTTF_{sys}$. Our first priority was to determine if there is significant advantage to this scheme which we believe has been answered in the affirmative in this paper. There does remain an interesting problem with respect to implementation of this scheme. For small values of G and C, it is not difficult to obtain the corresponding addresses for buddy and parity blocks. But for reasonable values of G and C the simple scheme of using all combinations of G disks out of C becomes unmanageable. For example for $G = 10$ and $C = 20$ there are $\binom{20}{10}$ different groups (times 10 to account for precessing the parity block). This is a large number for table lookup implementation to obtain addresses for buddy and parity blocks. To implement the various algorithms (e.g., to determine the address of the parity block

for a data block) the addresses of the buddy blocks must be either easily computable or the set of different groups should be small enough to allow for table lookup. This is still an open problem which we are currently investigating. It appears to be a combinatorial block design problem [2]. This paper addresses the issue of evaluating the proposed architecture under the assumption that the group assignment and addressing issues will be adequately solved given that we can show sufficient advantages to the larger cluster size.

2.2 Disk Rebuild Strategies.

In this section we describe several options that might be employed in rebuilding the failed disk. We start with a description of the simple baseline copy procedure and then discuss the more sophisticated schemes in following subsections.

An assumption common in all the rebuild schemes we consider concerns how writes to the failed disk are handled. It is clear that writes of blocks that have already been copied to the standby disk should be redirected to the standby to keep that portion of the disk up to date. For writes to the portion of the disk that has not yet been copied there are two possible options: (1) convert the write to a null operation or, (2) write the block to the corresponding block on the standby disk. We will assume in all schemes that all writes to the failed disk are redirected to the standby disk. This appears to be relatively easy to accomplish using a bit map to indicate which blocks have already been copied. The copy procedure will just skip over the blocks that have already been copied as a result of the write operations. The bit map would require no more than 16 Kbytes for reasonable block size and disk capacity.

Baseline Copy Procedure.

The baseline procedure then simply sequentially reads blocks from the failed disk (causing reconstruction) and writes them to the standby disk. The only exception to the sequential copying is to skip over blocks already copied due to writes in the normal workload.

Rebuild with Redirection of Reads.

When the failed disk has been partially reconstructed on the standby, it is possible to satisfy some reads by either reconstructing the block (by multiple reads of blocks on the surviving disks) or by reading from the standby (if the block has already been copied). Which of the two options is optimal will depend on whether the surviving disks are the bottleneck in the rebuild process or the standby disk is the bottleneck (and the optimum choice can change with time). Let $f(t)$ be the fraction of the failed disk that has been reconstructed at time t (the rebuild starts at $t = 0$). If we assume that the access pattern of the normal workload is uniform across all blocks, then the fraction of reads that can be redirected to the standby disk at time t can be denoted by $b(t) \cdot f(t)$ where $b(t)$ is a control variable which can vary between 0 and 1. In the analysis presented in the next section we determine the optimal policy and assume perfect adherence to that optimal policy.

Piggy-backing Rebuild on Normal Workload.

The idea here is to "capture" a block from the failed disk that is reconstructed due to a read request that was issued as part of the normal workload. A relatively simple implementation is possible assuming only a slight modification to a conventional buffer manager. When a read from the failed disk is satisfied by reconstruction of the block, the block contents will be stored in a buffer. At this time buffer copy can be marked as "modified", the disk address associated with the buffer page can be changed to refer to the standby disk, and the rebuild bit map can be changed to show this page as being copied. Normal buffer replacement will eventually copy the contents to the appropriate block on the standby. (At the end of the rebuild any blocks remaining in the cache can be flushed to the standby disk.)

Piggy-backing and redirection can help to reduce the load on the surviving disks in an array which has a failed disk. The extent to which this can be of benefit depends on several factors. For the same rebuild rate and the same minimum access rate for the normal workload, both redirection and piggy-backing can be used to reduce the load on the surviving disks. The I/O access capacity thus made available on the surviving disks can be used to either support a higher normal workload during rebuild (λ_0) or a faster rebuild. In the following section we present an analysis of disk arrays under failure to quantify the tradeoffs between (1) the cluster size C, (2) the minimum workload that must be supported λ_0 and, (3) the $MTTF_{sys}$. Qualitatively the tradeoffs are clear but quantitatively they are not. With a constant minimum workload λ_0, a larger cluster size will allow a faster copy rate but the larger cluster size increases the likelihood of a second failure in the cluster. Which effect is dominant is not immediately obvious. As we will show from the analysis in the next section, a larger cluster size can sometimes simultaneously support a higher minimum workload and provide a longer $MMTF_{sys}$.

3 Performance Analysis

We first summarize the notation used in the analysis:

μ	=	maximum capacity of a disk in access per second.
C	=	number of disks in a cluster.
N_c	=	number of clusters in the system.
N_{disk}	=	total number of disks in the system, which is equal to CN_c.
I	=	number of blocks on a disk.
G	=	the group size: $G-1$ data blocks and 1 parity block.
α	=	$(G-1)/(C-1)$, and $0 < \alpha \leq 1$
$\lambda(t)$	=	instantaneous throughput of a disk in accesses per second.
$\lambda_c(t)$	=	instantaneous copy rate in accesses per second.
λ_0	=	minimum throughput for normal workload in accesses per sec.
w	=	fraction of disk accesses in the normal workload that are writes.
ρ	=	fraction of disk accesses in the normal workload that are reads.
$b(t)$	=	the control variable for dynamic re-routing, $0 \leq b(t) \leq 1$.
$f(t)$	=	instantaneous fraction of the failed disk image that has been reconstructed by time t.

In this section, we derive expressions for the rebuild time and $MTTF_{sys}$ for the recovery techniques introduced in the previous section. We first outline the analysis and provide an intuitive interpretation. Since the analysis is similar in all cases, we present only the analysis of the *rebuild with redirection of reads* in detail. Results for the *Baseline copy procedure* and *Piggy-backing* are given in the appendix.

An informal explanation of the results is rather straightforward. In all cases there are two constraints that relate the copy rate $\lambda_c(t)$ and the normal workload, λ_0. In the case of rerouting there is also the control variable that can be used to shift some load between the surviving disks and the standby. Each of the two constraints can be viewed as providing an upper bound on $\lambda_c(t)$. Each of the constraints is also linear in λ_0, $f(t)$ and $b(t)$. The only complication is that the bounds shift with time as $f(t)$ (the portion of the disk that has been rebuilt) increases. At any time t, as $b(t)$ is varied from 0 to 1 each constraint equation sweeps out a region on the λ_0 - λ_c plane as shown in Figure 4[1]. Given a particular value of λ_0, say λ_0^*, the problem is to determine the value of $b(t)$ that maximizes $\lambda_c(t)$. It is not difficult to show that this will be the value of $b(t)$, say b^*, for which the two constraint equations give the same value for $\lambda_c(t)$. This is illustrated in Figure 4.

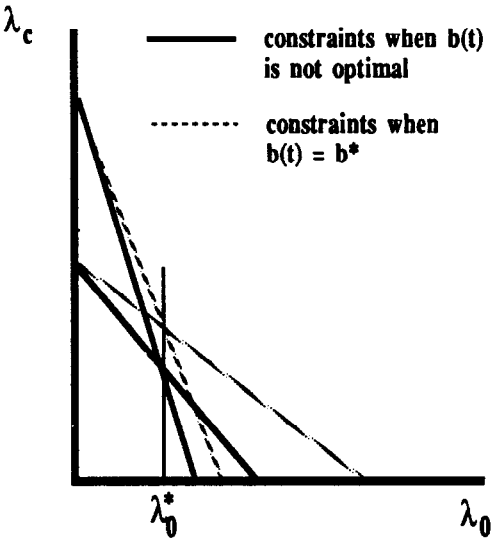

Figure 4: Varying feasible solution space as the control variable $b(t)$ varies.

Controlling the value of $b(t)$ can be viewed as attempting to maximize the instantaneous copy rate subject to the constraints. The two constraints reflect that neither of the two disks can be greater than 100% utilized. In practice, $b(t)$ can be adjusted heuristically based on measuring the utilization of the disks. If the surviving disks were the bottleneck and the re-route was not enabled, we should set the control variable $b(t) = 1$ and re-route the workload to the standby. If the standby disk is the bottleneck and the re-route operation was enable, we should shift some workload back to the surviving disks. The optimal control policy would use this degree of freedom to try to balance the load as much as possible and thereby maximizing the instantaneous rebuild rate.

3.1 Rebuild Time

In this section, we develop an expression for the rebuild time in the case of *Rebuild with Redirection of Reads*. The first constraint (C1 below) comes from considering the surviving disks in the array and the second constraint (C2) comes from considering the standby disk.

$$\mu \geq \lambda(t) + \alpha\rho\lambda(t)[1 - b(t)f(t)] + \alpha\lambda_c(t) \quad (C1)$$
$$\mu \geq \lambda_c(t) + w\lambda(t) + \rho\lambda(t)b(t)f(t) \quad (C2)$$

[1]For the Piggybacking strategy, it is possible to choose an optimal control such that the slope of both constraints will increase.

with :

$$f(t) = \frac{1}{I}\int_0^t \{\lambda_c(\tau) + w\lambda(\tau)[1 - f(\tau)]\}\, d\tau$$

For (C1), the first term is the regular workload to the surviving disk, the second term is the fraction of workload referencing the un-rebuilt portion of the failed disk and the last term is the copy workload to the surviving disk. For (C2), the first term is the copy workload, the second term reflects the updates to the failed disk and the last term is the fraction of the workload referencing the rebuilt portion of the failed disk at time t.

Figure 5 illustrates the feasible solution space during the rebuild period. It is important to note that the feasible solution space is time-varying and a higher instantaneous workload can be supported as more of the failed disk is rebuilt.

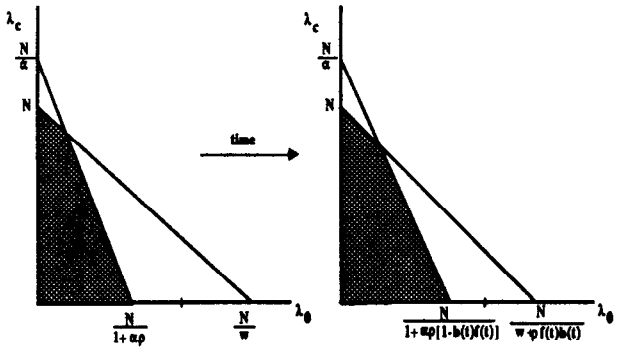

Figure 5: Feasible solution space for $\lambda_c(t)$ and $\lambda(t)$ during the rebuild period.

Depending on the value of the required minimum workload λ_0, there are two cases :

Case 1: $0 \le \lambda_0 \le \frac{\mu(1-\alpha)}{1+\alpha(\rho-w)}$.

In this case the optimal $\lambda_c(t)$ is obtained by saturating the standby disk. To maximize the copy rate, the control variable $b(t)$ should be set to 0 during for the entire rebuild period and thus during the rebuild period $\lambda_c^*(t)$ will remain constant. $\lambda_c^*(t)$ is:

$$\lambda_c^*(t) = \mu - w\lambda_0 \qquad (1)$$

Once we know $\lambda_c^*(t)$, we can substitute it into the expression of $f(t)$. Using Laplace Transform we can obtain $f^*(s)$, the transform of $f(t)$, and by inverse transform operation, we can obtain $f(t)$.

$$f^*(s) = \frac{\mu/I}{s(s + \frac{w\lambda_0}{I})}$$

$$f(t) = \frac{\mu}{w\lambda_0}\left[1 - e^{-\frac{w\lambda_0}{I}t}\right]$$

But equation $f(t) = 1$, we can find the rebuild time T which can be expressed as:

$$T = -\frac{I}{w\lambda_0}\ln\left(1 - \frac{w\lambda_0}{\mu}\right) \qquad (2)$$

Case 2: $\frac{\mu(1-\alpha)}{1+\alpha(\rho-w)} \le \lambda_0 \le \frac{\mu}{1+\alpha\rho}$.

There are two phases in the rebuild process. In phase 1, $\lambda_c^*(t)$ is chosen to saturate the surviving disks. During this phase the control variable $b(t)$ should be set to 1 to maximize the copy rate. As the failed disk is rebuilt, a time t_1 *may* be reached such that both the surviving disks and the standby disk disks are saturated. If this occurs before the rebuild is completed, then phase 2 begins. In phase 2 both the standby disks and the surviving disk will be saturated. To maximize the copy rate after t_1, the control variable $b(t)$ is set so that only those reads that reference the rebuilt portion of the failed disk by time t_1 will be rerouted.

To compute the optimal copy rate $\lambda_c^*(t)$ in phase 1, we let the control variable $b(t) = 1$. Applying Laplace to $f(t)$ and substituting the transform $f^*(s)$ into the Laplace transform of (C1) and by inverse transform operation, the optimal copy rate $\lambda_c^*(t)$ is obtained as:

$$\lambda_c^*(t) = \frac{-(\mu-\lambda_0)w}{\alpha(\rho-w)} + \left[\frac{\rho(\mu-\lambda_0) - \rho\lambda_0\alpha(\rho-w)}{\alpha(\rho-w)}\right] e^{\frac{\lambda_0}{I}(\rho-w)t} \qquad (3)$$

Then $\lambda_c^*(s)$ (the Laplace Transform of $\lambda_c^*(t)$) can be substituted into the expression of $f^*(s)$ and obtain $f(t)$, which is :

$$f(t) = \left[\frac{(\mu-\lambda_0)}{\alpha(w-\rho)\lambda_0} + 1\right]\left[1 - e^{\frac{-\lambda_0}{I}(w-\rho)t}\right]$$

To find the rebuild time, let $f(t) = 1$ and we find the time t^* at which $f(t) = 1$:

$$t^* = \frac{-I}{\lambda_0(w-\rho)}\ln\left[1 - \frac{\alpha\lambda_0(w-\rho)}{\mu - \lambda_0 + \lambda_0\alpha(w-\rho)}\right]$$

However, the rebuilding process may not be completed by time t^* since by time t_1, both the surviving disk and the standby disks are saturated. To obtain t_1, we can equate the two constraints (C1) and (C2). Since $\lambda_c(t) = \lambda_c^*(t)$ and by time t_1, $\lambda(t) = \lambda_0$. t_1 can be expressed as:

$$t_1 = \frac{I}{\lambda_0(\rho-w)}\ln\left[\frac{K_1}{K_2}\right] \qquad \textit{where}$$

$$K_1 = \frac{(1+\alpha)(\mu-\lambda_0)}{2\alpha} + \frac{(\mu-\lambda_0)w}{\alpha(\rho-w)} \qquad \textit{and}$$

$$K_2 = \frac{(\mu-\lambda_0)\rho}{\alpha(\rho-w)} - \rho\lambda_0$$

Therefore, if $t^* \leq t_1$, the rebuilding is finished at t^*, otherwise phase 2 of the rebuilding process starts at time t_1. To maximize the copy rate in phase 2, only the reads which reference the rebuilt data before t_1 are redirected to the standby disk. The optimal copy rate after t_1 will be constant and is equal to:

$$\lambda_c^*(t) = \mu - \lambda_0[w + \rho f(t_1)] \quad for\ t \geq t_1 \quad (4)$$

From $\lambda_c^*(t)$ where $t \geq t_1$, we can obtain the time t_2, to complete the rebuilding process:

$$t_2 = \frac{-I}{\lambda_0 w} \ln \left[1 - \frac{[1 - f(t_1)]w\lambda_0}{\lambda_c^*(t_1) + w\lambda_0[1 - f(t_1)]} \right]$$

Therefore, the rebuild time T will be :

$$T = \begin{cases} t^* & if\ t^* \leq t_1 \\ t_1 + t_2 & if\ t_1 > t^* \end{cases} \quad (5)$$

The optimal copy rate and rebuild time of the baseline recovery procedure and piggy-backing recovery procedure are summarized in the appendix.

3.2 Mean Time to System Failure ($MTTF_{sys}$)

For a RAID system, whenever two or more disks fail in the same cluster, data will be unavailable and we will consider that the system has failed. If this situation occurs, it is assumed that a checkpoint-and-log recovery technique is initiated to recover the data.

To calculate the $MTTF_{sys}$, we assume the failure times of each disk are identically distributed and independent exponential random variables. Let $MTTF_{disk}$ be the mean time to failure for a single disk and let T be the rebuild time. Let $MTTF_{cluster}$ be the mean time to data unavailability for a cluster, then $MTTF_{cluster}$ can be expressed as:

$$MTTF_{cluster} = \frac{MTTF_{disk}}{C} * \frac{1}{1 - e^{-\frac{(C-1)T}{MTTF_{disk}}}} \approx \frac{(MTTF_{disk})^2}{C(C-1)T}$$

$MTTF_{sys}$ can then be expressed as:

$$MTTF_{sys} = \frac{MTTF_{cluster}}{N_c} = \frac{C(MTTF_{cluster})}{N_{disk}} \approx \frac{(MTTF_{disk})^2}{N_{disk}(C-1)T} \quad (6)$$

Since $\alpha = (G-1)/(C-1)$, we can also express $MTTF_{sys}$ as:

$$MTTF_{sys} \approx \frac{\alpha(MTTF_{disk})^2}{N_{disk}(G-1)T} \quad (7)$$

We note that $MTTF_{sys}$ is inversely proportional to the cluster size and also inversely proportional to the disk rebuild time.

4 Discussion

Figures 6 ,7 and 8 show the $MTTF_{sys}$ as a function of α and for different values of λ_0. In all cases the group size is fixed at $G = 5$, the total number of disks in the system is 1000 and $MTTF_{disk} = 30,000$ hours. The three figures differ in the fraction of reads versus writes in the normal workload.

Consider first Figure 6 in which the workload is 50% reads and 50% writes. The three graphs (from top to bottom) correspond to (1) the baseline rebuild policy, (2) the strategy with dynamic re-routing and (3) the strategy with piggy-backing and re-routing. It is easy to see that for any particular values for α and λ_0 the $MTTF_{sys}$ is increased and sometimes by as much as 50%. The original RAID architecture with $\alpha = 1$ corresponds to the points along the vertical axis to the far right. From these points one can see that with $\alpha = 1$ a workload of $\lambda_0 = 35$ can not be supported regardless of the rebuild policy used. However, in a clustered RAID with $\alpha \leq 0.7$ (i.e., $C \geq 7$) and utilizing the more sophisticated rebuild strategy, a workload of 35 accesses per second can be supported. From the bottom graph we see that the $MTTF_{sys}$ is reduced to approximately 14 years to support this workload. Using this same example we can see the beneficial effect of the more sophisticated rebuild schemes. For a workload of 35 accesses per second and $\alpha = 0.7$ the $MTTF_{sys}$ goes from 2 years for the baseline rebuild strategy to 14 years for the piggy-back and re-route strategy.

The graph corresponding to the baseline rebuild strategy shows that as α decreases (i.e., the cluster size increases) the $MTTF_{sys}$ can actually increase. For this case the increased rate at which a second failure occurs in the cluster is more than compensated for by a decreased rebuild time (at least over some range of values for α). For small enough α, the $MTTF_{sys}$ eventually starts to decrease. This occurs because the standby disk becomes the bottleneck and decreasing α further cannot speed the rebuild process but it does increase the rate of a second failure in the cluster. For the other two rebuild schemes the $MTTF_{sys}$ is flat at

first and then monotonically decreasing with decreasing α. In the flat part the decreased rebuild time is almost exactly compensated for by the increased rate of a second failure. As in the baseline case a large enough cluster will eventually saturate the standby disk and the $MTTF_{sys}$ will then start to fall.

Figure 7 illustrates a 100 percent read workload. It has the same general behavior as in Figure 6 except that all of the characteristics are more pronounced. A pure read case is the worst case in terms of the reconstruction overhead per access in the regular workload. Thus for the same λ_0 there is more load per surviving disk and more of an impact on the rebuild time. In this situation the re-routing and piggy-backing rebuild strategies have a proportionately bigger effect. For example, for a required $MTTF_{sys} = 10$ years, with the baseline strategy, $\lambda_0 \leq$ approximately 27. With the re-routing and piggy-backing rebuild strategy λ_0 can be approximately 36, an increase of 33%.

In Figure 8 the same behavior is again illustrated for a 100% write workload. This type of workload imposes the least reconstruction overhead penalty. Just as the worst case (a pure read workload) showed more pronounced benefit from the larger cluster sizes and the more sophisticated rebuild strategies, the pure write workload shows the least benefit. In this case if we required a $MTTF_{sys}$ of at least 10 years, then with the baseline rebuild strategy and $\alpha = 0.30$ we can have $\lambda_0 = 35$ (approximately). With re-routing and piggy-backing we can support $\lambda_0 = 40$ (approximately) which is an increase of only 15%.

For the analysis in the previous section, we assume an uniform distribution in accessing the data. In reality, the distribution of accessing the data is often non-uniform. In this section, we assume a $80 - 20$ model in which 80 percent of the disk accesses are to 20 percent of the data in the database.

We used simulation to study the impact of the non-uniform access pattern and compare the $MTTF_{sys}$ with the uniform reference case. Figure 9 shows the $MTTF_{sys}$ for the uniform and $80 - 20$ access pattern. Under low to medium require workload λ_0, the $MTTF_{sys}$ of the uniform access case is only at most 6 percent higher than the $MTTF_{sys}$ of the non-uniform case. However, when the workload is write-intensive as depicted in Figure 10, $MTTF_{sys}$ of the uniform access case can be up to 20 percent higher than the skewed access case. This is due to a large part of the standby disk throughput being used for writes in the normal workload.

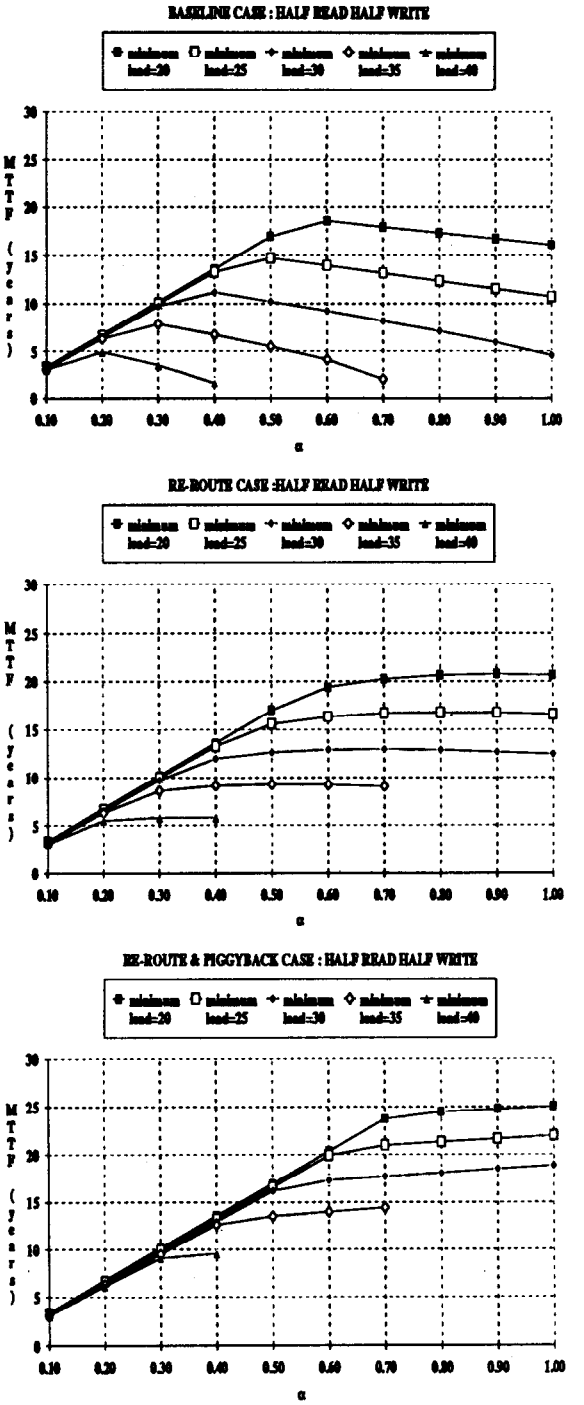

Figure 6: $MTTF_{sys}$ for various recovery scheme for half read half write transactions.

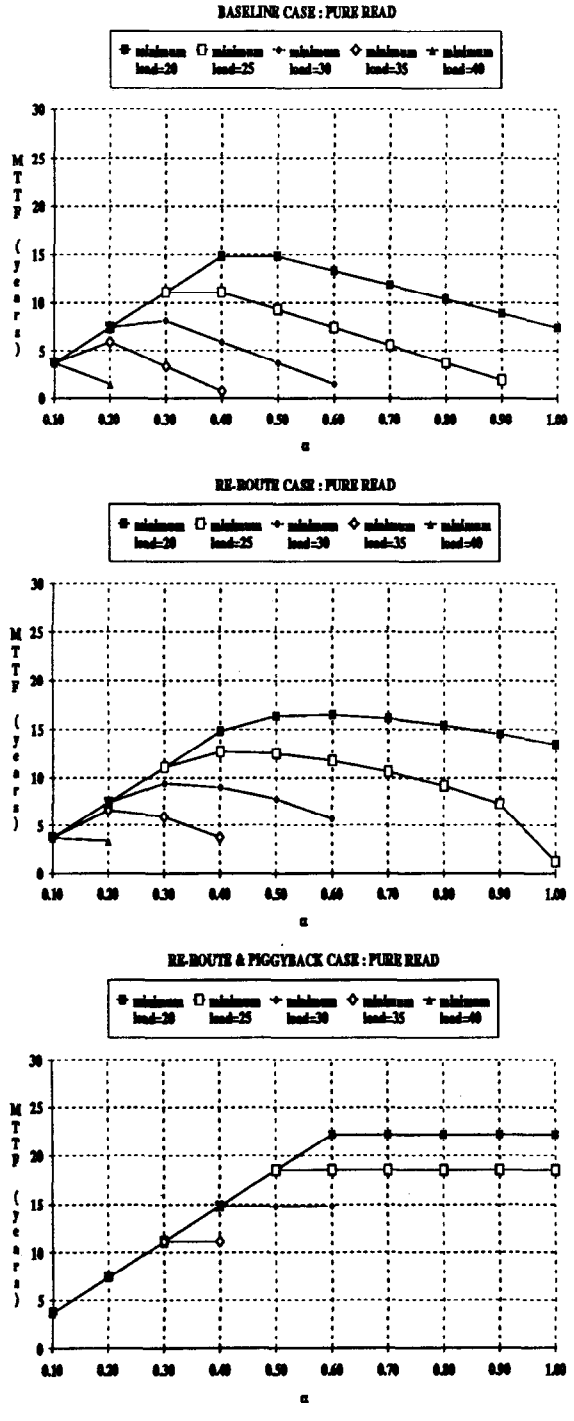

Figure 7: $MTTF_{sys}$ for various recovery scheme for pure read transactions.

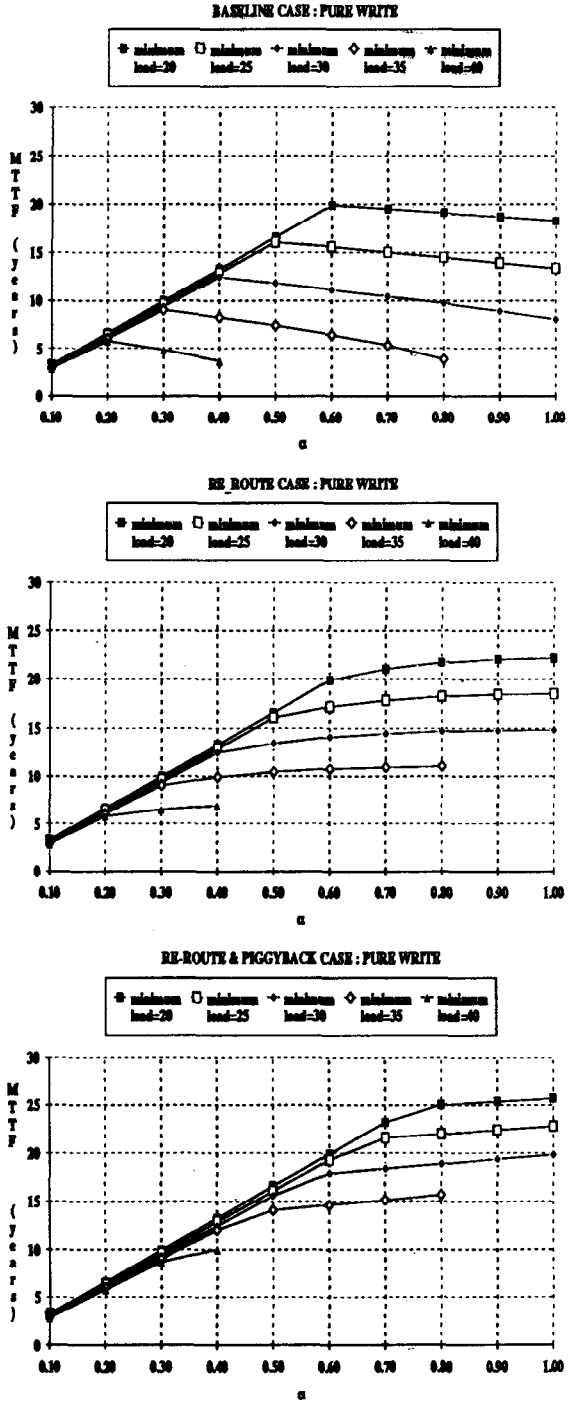

Figure 8: $MTTF_{sys}$ for various recovery scheme pure write transactions.

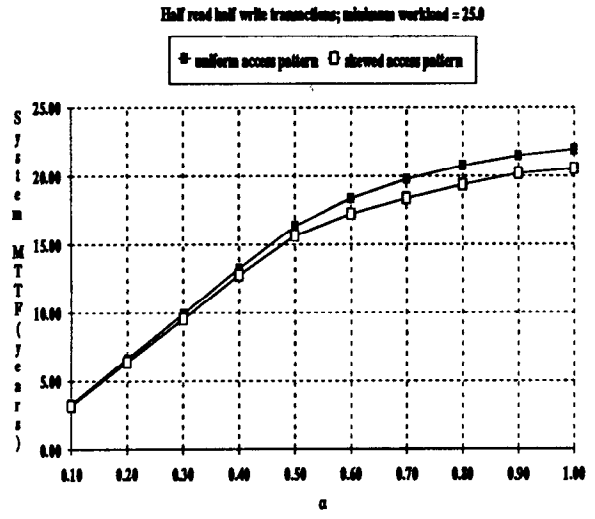

Figure 9: Comparison of rebuild time for uniform access vs. skewed access under light load.

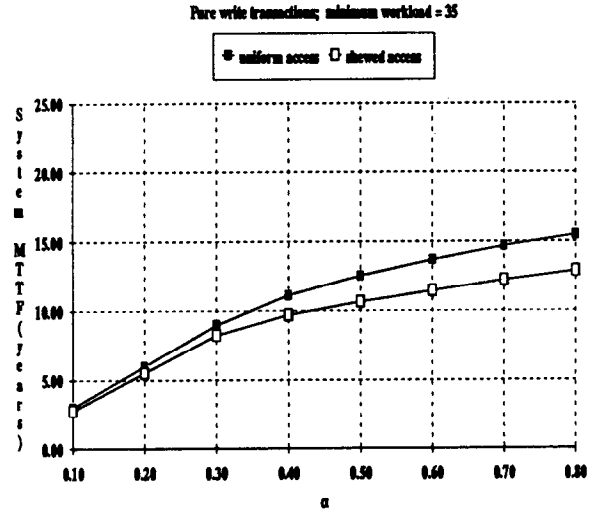

Figure 10: Comparison of rebuild time for uniform access vs. skewed access under write intensive workload.

5 Conclusion.

The $N+1$ RAID architecture in which the group size is equal to the cluster size is in the worst case, not able to support a workload of more than 50% load on the surviving disks. This worst case is a workload of all reads. In other cases the degradation is not as severe but can still be significant. A clustered RAID organization has been proposed in which data plus parity groups are distributed over a larger number of disks such that, when there is a failure, the increased load per disk is reduced. A larger cluster size can potentially have a negative effect in that the rate of having a second failure in a cluster (and thus data becomes unavailable) can increase. The analysis reported in this paper shows that a larger cluster size can be used to advantage to speed the rebuild of the failed disk and also to support a larger workload during the rebuild period. The tradeoffs between the workload that is supported during rebuild, the cluster size and the mean time to system failure were quantified by the analysis and illustrated by numerical examples. Several rebuild schemes were defined and analyzed. The more sophisticated schemes were demonstrated to have significant potential benefit, particularly in conjunction with the larger cluster sizes.

An open problem with respect to the clustered RAID architecture is the issue of mapping the address of a data block to the addresses of the parity block and the other data blocks in the same group. A scheme which provides the symmetry to balance the load over the surviving disks in a cluster and also to provide a practical address mapping function is a subject of current research.

References

[1] P. Chen. An evaluation of redundant arrays of disks using an Amdahl 5890. Technical report, UC Berkeley, 1989.

[2] C.J. Colbourn and P.C. van Oorshot. Applications of combinatorial designs in computer science. *ACM Computing Surveys*, 21:223–250, 1989.

[3] G. Copeland and T. Keller. A comparison or high-availability media recovery techniques. In *Proceedings SIGMOD 1989*, pages 98–109, 1989.

[4] G.A. Gibson. Performance and reliability in redundant arrays of inexpensive disks. In *1989 CMG Annual Conference Proc.*, 1989.

[5] H.I. Hsiao and D.J.DeWitt. Chained declustering: A new availability strategy for multiprocessor database machines. In *Sixth Int'l Conf on Database Engineering*, pages 456–465, 1990.

[6] M. S. Lakshmi and P. S. Yu. Effect of skew on join performance in parallel architectures. In

Int. Symposium on Databases in Parallel and Distributed Systems, pages 107–120, 1988.

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

[8] M. Stonebraker and G.A. Schloss. Distributed raid - a new multiple copy algorithm. In *Sixth Int'l. Conf on Data Engineering*, pages 430–437, 1990.

Appendix

A Results of the Baseline Recovery Procedure

The constraints for the surviving and standby disks are :

$$\mu \geq \lambda(t) + \alpha\rho\lambda(t) + \alpha\lambda_c(t) \quad (C1)$$
$$\mu \geq \lambda_c(t) + w\lambda(t) \quad (C2)$$

with :

$$f(t) = \frac{1}{I}\int_0^t \{\lambda_c(\tau) + w\lambda(\tau)[1 - f(\tau)]\}\, d\tau$$

In (C1), the first term on the right is the regular workload on a surviving disk, the second term is the normal workload resulting from reads to the failed disk, and the last term is the copy workload to the surviving disk. For (C2), the first term is the copy workload and the second term reflects the updates to the failed disk.

Depending on λ_0, the optimal copy rate $\lambda_c^*(t)$ is :
if $0 \leq \lambda_0 \leq \frac{\mu(1-\alpha)}{1+\alpha(\rho-w)}$:

$$\lambda_c^*(t) = \mu - w\lambda_0 \quad (8)$$

if $\frac{\mu(1-\alpha)}{1+\alpha(\rho-w)} \leq \lambda_0 \leq \frac{\mu}{1+\alpha\rho}$:

$$\lambda_c^*(t) = \frac{\mu - \lambda_0[1+\alpha\rho]}{\alpha} \quad (9)$$

The rebuild time T can be expressed as:
if $0 \leq \lambda_0 \leq \frac{\mu(1-\alpha)}{1+\alpha(\rho-w)}$:

$$T = -\frac{I}{w\lambda_0}\ln\left(1 - \frac{w\lambda_0}{\mu}\right) \quad (10)$$

if $\frac{\mu(1-\alpha)}{1+\alpha(\rho-w)} \leq \lambda_0 \leq \frac{\mu}{1+\alpha\rho}$:

$$T = -\frac{I}{w\lambda_0}\ln\left(1 - \frac{w\lambda_0}{\left[\frac{\mu-\lambda_0(1+\alpha\rho)}{\alpha} + w\lambda_0\right]}\right) \quad (11)$$

B Results of the Piggybacking Recovery Procedure

Again the two constraints $(C1)$ and $(C2)$ are :

$$\mu \geq \lambda(t) + \alpha\rho\lambda(t)[1 - b(t)f(t)] + \alpha\lambda_c(t)$$
$$\mu \geq \lambda_c(t) + w\lambda(t) + \rho\lambda(t)b(t)f(t) + \rho\lambda(t)[1 - f(t)]$$

with :

$$f(t) = \frac{1}{I}\int_0^t \{\lambda_c(\tau) + w\lambda(\tau)[1 - f(\tau)] + \rho\lambda(\tau)[1 - f(\tau)]\}\, d\tau$$

These constraints have the same interpretation as the *re-route recovery procedure* except the last term in (C2) and $f(t)$ represents the piggy-backing workload.

Depending on the λ_0, we have the optimal copy rate $\lambda_c^*(t)$ and rebuild time T :

Case 1 : If $0 \leq \lambda_0 \leq \frac{\mu(1-\alpha)}{(1+2\alpha\rho-\alpha)}$.

$$\lambda_c^*(t) = -\lambda_0 + \frac{\mu}{1-\rho}\left[1 - \rho e^{-\frac{\lambda_0}{I}(1-\rho)t}\right] \quad (12)$$

$$T = -\frac{I}{w\lambda_0}\ln\left(1 - \frac{w\lambda_0}{\mu}\right). \quad (13)$$

Case 2 : If $\frac{\mu(1-\alpha)}{(1+2\alpha\rho-\alpha)} \leq \lambda_0 \leq \frac{\mu(1-\alpha)}{[1+\alpha(\rho-1)]}$.
for $t \leq t_1$:

$$\lambda_c^*(t) = -\lambda_0 + \frac{\mu}{1-\rho}\left[1 - \rho e^{-\frac{\lambda_0}{I}(1-\rho)t}\right] \quad (14)$$

for $t > t_1$:

$$\lambda_c^*(t) = \left(\frac{2K + \lambda_0\rho[1 - f(t_1)]}{2-\rho}\right) - \left(\frac{K\rho + \lambda_0\rho[1 - f(t_1)]}{2-\rho}\right) e^{-\frac{\lambda_0}{I}(1-\rho/2)(t-t_1)} \quad (15)$$

where :

$$f(t_1) = \frac{\mu}{\lambda_0(1-\rho)}\left[1 - \frac{1}{\rho} + \frac{\mu - \lambda_0(1+\alpha\rho) + \alpha\lambda_0}{\alpha\rho\mu}(1-\rho)\right]$$

$$K = (\mu - \lambda_0)(1 - \frac{\alpha-1}{2\alpha}) - \frac{1}{2}\lambda_0\rho + \frac{1}{2}\rho\lambda_0 f(t_1)$$

$$t_1 = -\frac{I}{\lambda_0(1-\rho)}\ln\left[\frac{1}{\rho} - \frac{\mu - \lambda_0(1+\alpha\rho) + \alpha\lambda_0}{\alpha\rho\mu}(1-\rho)\right]$$

The rebuild time T will be :

$$T = t_1 + \frac{I}{\lambda_0(\frac{\rho}{2}-1)} \ln\left[\frac{C_1 - 1 + f(t_1)}{C_2}\right] \quad (16)$$

where :

$$C_1 = \frac{2K}{\lambda_0(2-\rho)} + \frac{\rho[1-f(t_1)]}{2-\rho} + 1 - f(t_1)$$

$$C_2 = \frac{2K}{\lambda_0(2-\rho)} + \frac{2[1-f(t_1)]}{2-\rho}$$

Case 3: if $\frac{\mu(1-\alpha)}{1+\alpha(\rho-1)} \leq \lambda_0 \leq \frac{\mu}{1+\alpha\rho}$.
for $t^* \leq t_1$:

$$\lambda_c^*(t) = \frac{\mu-\lambda_0}{\alpha(1-\rho)} - \left[\frac{(\mu-\lambda_0)\rho}{\alpha(1-\rho)} + \rho\lambda_0\right] e^{-\frac{\lambda_0}{I}(1-\rho)t} \quad (17)$$

for $t^* > t_1$:

$$\lambda_c^*(t) = \left(\frac{2K + \lambda_0\rho[1-f(t_1)]}{2-\rho}\right) - \left(\frac{K\rho + \lambda_0\rho[1-f(t_1)]}{2-\rho}\right) e^{-\frac{\lambda_0}{I}(1-\rho/2)(t-t_1)} \quad (18)$$

where :

$$f(t_1) = \frac{\mu - \lambda_0 + \alpha\lambda_0(1-\rho)}{\alpha\lambda_0(1-\rho)}\left[1 - e^{-\frac{\lambda_0}{I}(1-\rho)t_1}\right]$$

$$K = (\mu-\lambda_0)(1-\frac{\alpha-1}{2\alpha}) - \frac{1}{2}\lambda_0\rho + \frac{1}{2}\rho\lambda_0 f(t_1)$$

$$t^* = -\frac{I}{\lambda_0(1-\rho)} \ln\left[1 - \frac{\alpha\lambda_0(1-\rho)}{\mu - \lambda_0 + \alpha\lambda_0(1-\rho)}\right]$$

$$t_1 = -\frac{I}{\lambda_0 w} \ln\left[\frac{(\mu-\lambda_0)(1-\alpha w)}{(\mu-\lambda_0)\rho + \alpha\rho\lambda_0 w}\right]$$

The rebuild time T will be :
If $t^* \leq t_1$:

$$T = t^* \quad (19)$$

If $t^* > t_1$:

$$T = t_1 + \frac{I}{\lambda_0(\rho/2-1)} \ln\left[\frac{C_1 - 1 + f(t_1)}{C_2}\right] \quad (20)$$

where :

$$C_1 = \frac{2K}{\lambda_0(2-\rho)} + \frac{\rho[1-f(t_1)]}{2-\rho} + 1 - f(t_1)$$

$$C_2 = \frac{2K}{\lambda_0(2-\rho)} + \frac{2[1-f(t_1)]}{2-\rho}$$