UNIVERSITY OF CALIFORNIA, SAN DIEGO

Reliability and Performance of Disk Arrays

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy
in Computer Science

by

Thomas Johannes Emil Schwarz

Committee in charge:

Professor Walther A. Burkhard, Chairman
Professor Ting Ting Lin
Professor Christos H. Papadimitriou
Professor Joseph C. Pasquale
Professor Jack K. Wolf

1994

The dissertation of Thomas Schwarz is approved, and it is acceptable in quality and form for publication on microfilm:

_____

_____

_____

_____

_____

Chair

University of California, San Diego

1994

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

VITA

Dec. 15, 1955	Born, Bonn, Federal Republic of Germany
1974-80	Studies of Mathematics at Westphälische Wilhelms-Universität Münster
1980	Diplom Mathematiker (Diploma in Mathematics), Westphälische Wilhelms-Universität Münster with a Minor in Business Administration. The thesis title was: *On Maximal Involution Invariant Orders in Simple Algebras over the Quotient Field of a Dedekind Ring.*
1980-84	Scientific Employee at Fernuniversität Hagen
1984	Doctorate in Mathematics, Fernuniversität Hagen. The thesis title was *Simple Jordan Triple Systems.* The thesis advisor was Prof. Holger Petersson.
1984-85	Visiting Assistant Professor of Mathematics at Western Washington University, Bellingham, Washington
1985-87	Assistant Professor of Mathematics at Ohio State University, Mansfield and Columbus, Ohio
1987-88	Assistant Professor of Mathematics at California State University, Bakersfield, California
1988-91	Graduate Student in Computer Science, University of California at San Diego
1991-93	Jesuit Novitiate
1993-94	Visiting Assistant Professor of Computer Science (part-time), Gonzaga University, Spokane, Washington
1993-94	Graduate Student in Computer Science, University of California at San Diego

Publications:

1. Automorphism Groups of Simple Jordan Pairs,

   Algebras, Groups and Geometries **1** (1984), 490-509

2. Special Simple Jordan Triple Systems

   Algebras, Groups and Geometries **2** (1985), 117-128

3. *With Tudor Zamfirescu:* Typical Convex Sets of Convex Sets,

   Journal of the Australian Mathematical Society, Series A **43** (1987), 287-290

4. Simple Exceptional 16-dimensional Jordan Triple Systems

   Proceedings of the American Mathematical Society **100** (1987), 623-626

5. Simple Exceptional Jordan Triple Systems in Characteristic 2

   Communications in Algebra **16**.11 (1988), 2247-2257

6. Triple Systems with Composition Forms

   Communications in Algebra **16**.12 (1988), 2569-2577

7. Tangeability of Homogeneous Convex Cones

   Algebras, Groups and Geometries **4** (1987), 451-457

8. *With Robert Bowdidge and Walter A. Burkhard:*

   Low Cost Comparison of Large Files

   Proceedings of the Tenth International Conference on Distributed Computing Systems, Paris, 1990, 192-202

9. *With Walter A. Burkhard and Kimberly C. Claffy:*

   Balanced Disk Array Schemes

   Eleventh IEEE Symposium on Mass Storage Systems, Monterey, CA, 1991, 45-50

10. *With Walter A. Burkhard:*

    RAID Organization and Performance

    Proceedings of the Twelfth International Conference on Distributed Computing Systems, Yokohama, 1992, 318-325

FIELDS OF STUDY

Major Field: Computer Science

Studies in Theoretical Computer Science.
Professors Patrik Dymond, Christos H. Papadimitriou, Michael Sacks and Heather Woll

Studies in Computer Systems.
Professors Laurette Bradley, Joseph C. Pasquale, George Polyzos and Augustus C. Uht

Studies in Operating Research.
Professor T. C. Hu

ABSTRACT OF THE DISSERTATION

Reliability and Performance
of Disk Arrays

by
Thomas Johannes Emil Schwarz
Doctor of Philosophy in Computer Science
University of California, San Diego, 1994
Professor Walther A. Burkhard, Chair

Introduction of recundancy into secondary storage systems in the form of Redundant Arrays of Independent Disks (RAIDs) has generated both research activity and commercial applications. The thesis investigates the performance and reliability of RAIDs analytically. In addition to Level 5 RAIDs with and without spares, it treats higher redundancy extensions of the Level 5 RAID: two-dimensional RAIDs and disk arrays based on Maximum Distance Separable (MDS) codes. Finally, we extend the use of MDS codes to implement a storage space saving distributed database.

Chapter 1

Introduction

Disk array storage systems, especially those with redundant arrays of independent disks (RAID) Level 5 data organization [28], provide excellent cost, run-time performance as well as reliability and will meet the needs of computing systems for the immediate future. Computing systems, especially those with massive storage requirements, may need even greater reliability than provided by RAID Level 5 (see [6].)

The reliability of higher level RAID organizations is gained through the introduction of redundancy by clustering data storage elements into reliability groups.

Our goal is to present and analyze data organization schemes for RAIDs including RAID Level 5 and schemes that provide higher levels of redundancy. The data organizations, derived from maximal distance separable (MDS) codes, retain some of the performance advantages of RAID Level 5 while providing higher reliability[33]. Gibson et al. present the *multi-dimensional* parity schemes [12] which form the basis for the other data organizations discussed here. Our data organizations incorporate novel combinations of spare disks [22] and strings[11]. We compare these schemes against each other and investigate the impact of two level of clustering (see [25]).

The thesis is organized as follows: Chapter 2 outlines the modeling assumptions and the tools used. We first give an overview of the Markov chain approach and then presents in detail the mathematical background needed for our calculations. The latter part is technical and the reader need not understand the calculations for the remainder of the report. We outline the basic components of the disk array and our modeling assumptions. Chapter 3 analyzes several varieties of Level 5 RAIDs; we investigate the effects of

a load balancing scheme and the effects of various ways to include spare disk space in the organization: designated spares, distributed sparing of a string and of a few disks, the latter in two flavours. We first present the organization and results, then we discuss the technical details of our investigation. Chapter 4 treats RAID designs based on MDS codes. We give a short introduction into the coding theoretical background and then analyze the reliability of some RAID designs based on them. Chapter 5 investigates the reliability of two dimensional RAID schemes. We first explore the potential of the scheme by taking only disk failures into account. We then examine the possible assignements of disks to strings. We give reliability numbers for the best stringing schemes. Chapter 6 describes implementation strategies for the "almost complete address translation scheme" (ACATS). Chapters 7 - 11 treat the performance of a RAID under non-failure mode using a variety of synchronization schemes and establish that the use of a non-volatile storage cache is imperative for good performance. Chapter 11 treats in detail the performance of the RAID under the most common failure modes and for all previously considered organizations. Chapter 12 finally gives an application of MDS codes to Distributed Data Bases and in particular discusses the extension of signature schemes to these schemes.

Chapter 2

Modeling Assumptions and Tools

2.1 Terminology

We begin by briefly reviewing the terminology we will be using throughout the paper.

A *disk array* or *RAID (Redundant Array of Independent Disks)* is a collection of independent disks, on which data is stored redundantly, such that failure of a limited number of components does not result in data loss.

A reliability group is a group of data storage elements, either whole disks or disk tracks, that are associated by shared data redundancy, such that failure of one (or more, depending on the scheme) element(s) leaves all the data accessible. We obtain the redundancy by storing in addition to the data so-called check data on one or more disks or disk tracks. For clarity, we refer to the original data as "message data."

A RAID Level 5 data organization reliability group contains data disks and a single parity check disk [28]; an MDS data organization reliability group contains data disks and a pair of check disks[33]. A two-dimensional parity RAID groups every storage element in two different reliability groups. Level 5 organizations can tolerate a single disk failure while the two dimensional disk array organization withstands a pair of concurrent disk failures and an MDS based array can be made to withstand even more.

Our orthogonal organization of strings and reliability groups is similar to that of Gibson [12]. A *string* is a group of disks that share hardware components such as power supply and cabling, cooling, small computer system interface (SCSI) controller and cabling

and host bus adapter (HBA) [11]. We consider three varieties of hardware redundancy within strings. A string is *soft* if it contains only the basic set of hardware components; as an example, a string with 10 disks would have two sets of HBAs and SCSI controllers, one power supply and one cooling fan. A string is *hardened* if some components are duplicated. Within this paper, our hardened strings will double the power supply, SCSI controllers and the HBAs components. A string is *super-hardened* if all the hardware components are duplicated. For us a super-hardened string will be a hardened string with duplicate cooling fans.

A very efficient method to extend the longevity of stored data is to provide spare disk space. In case of a disk or string failure, data stored on the failed device is reconstructed on stored in the spare space. We can have a single disk worth of spare space up to several strings of spare disks. The spare space can be provided by explicit spare disks or through distributed sparing, in which every disk contains some spare space. The advantage of distributed sparing consists in reducing the load at all disks in use. We will also investigate two distributed sparing schemes that only provide a disk or a few disks' worth of spare space to protect only against disk failure.

We can give the appearance of unlimited spare strings by quick repair of RAIDs, which replaces lost devices and causes the RAID to regenerate to the prefailure state. While short repair times (of a few hours) are a very effective data protection tool, the hidden (personal) costs can be formidable. We will assume rather leisurely repair times corresponding to repair on the next business day. Abundance of spare parts does not paly an important part.

Load balancing (among disks) is an important performance booster, especially in the presence of failed components. Distributed sparing is one instance in which we see performance benefits in the normal state. Muntz and Lui have made the observation that larger clusters than reliability groups are beneficial during the process of reconstruction on spares ([25].) We adjust their observation to RAIDs consisting of strings of disks. we introduce a logical and a physical disk layer in the RAID. The physical layer consists of the disks themselves whereas the logical layer superimposes a large number of virtual RAIDs over the physical layer. *Address translation* provides the connection between the layers. It translates the disk and track addresses of the virtual disks in the logical layer to disk

and track addresses of the physical layer. The virtual disks are arranged in fixed reliability groups with one virtual disk being the check disk and - in distributed sparing - another virtual disk being the spare. The easiest address translation scheme uses none: the logical and the physical layers are identical and the RAID is a level 4 RAID. The classic level 5 RAID uses address translation within the reliability group only. All blocks on a physical disk belong to one and the same reliability group, but the nature of the data stored on it varies. *Complete Address Translation* assigns virtual disk blocks to physical ones quasirandomly. The most interesting address translation scheme is the *Almost Complete Address Translation Scheme* (ACATS,) which respects the string structure of the RAID. It uses tracks as the basic unit of address translation to gain some performance benefits in the failure situation. ACATS translates a virtual track address, consisting of a virtual disk address and a track address. The virtual disk address consists of a string address and a disk-in-string address. The scheme then selects the physical string and then the physical disk-in-string address in a quasirandom way. Thus, it implements and address permutation of strings followed by a permutation of disk addresses in the string.

2.2 Markov Models

2.2.1 Overview

We will use Markov models to calculate the reliability of various disk array designs. We first give a general overview of the method used, the following sections will then give the calculations and deductions in more detail.

We capture the various failures modes in a RAID through a number of states. At each given moment, the RAID is in exactly one state. State transitions describe the failure of components and the repair of components. We always assume that there is only one transition at any time. If the probability, with which a state transition occurs, only depend on the current state, then the system is described in a Markov model. Our use of a Markov model is justified if we can assume that the failure and repair probabilities are memory-less, that is, that failure and repair probabilities are described by the exponential probability distribution. While this assumption conflicts with the proposed bath-tub probability (see fig. 2.2.1) for the failure probability of a RAID component or with the "bad batch phenomenon" in which disks failure depends on the production batch, the results will depend on the component MTTF (Mean Time To Failure) more than on the shape of the particular distribution and still offer excellent comparison material.

We are calculating MTTDL (Mean Time To Data Loss.) We describe data loss as transition to an absorbing "Failure" state, from which no repair is possible. All other states represent situations, in which all data stored in a RAID is accessible, possibly through several reconstruction processes, in which we exploit the intrinsic storage redundancy.

We define the reliability vector $\wp(t)$, whose dimension is the number of non-failure states and whose coefficients are the probabilities that the RAID is in the corresponding non-failure state at time t. The survival probability $p(t)$ of the array is simply the sum of the coefficients of this vector. The marginal probabilities for state transitions between non-failure states (i.e. the state transition rates) form the coefficients of the matrix M, whose diagonal elements are the negative probabilities that a state transition will lead away from a given state. In other words, M is defined by the differential equation

$$\frac{\delta \wp}{\delta t} = M \cdot \wp.$$

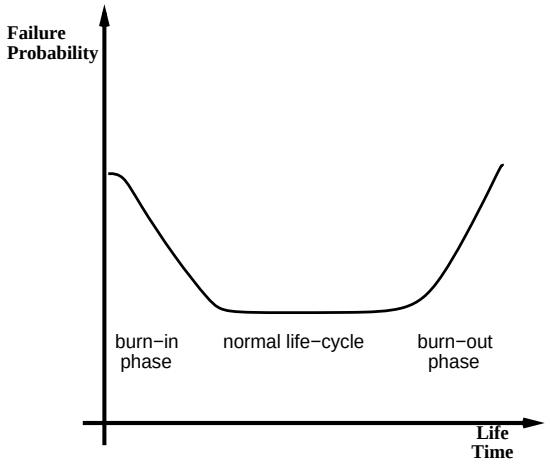


Figure 2.1: The Bath-Tub Probability Distribution for Component Failure

MTTDL is the expected value of the time to failure:

$$\text{MTTDL} = \int_0^\infty p(t)dt.$$

We use the Laplace transform to find an closed form expression for MTTDL:

$$\text{MTTDL} = -(1, 1, \ldots, 1) \cdot M^{-1} \cdot \wp(0)$$

where $\wp(0)$ is the initial state $(1, 0, \ldots, 0)$, that is, the normal state.

A sometimes useful reverse approximation is given by

$$p(t) = \exp(-t/\text{MTTDL}).$$

In the following sections we extend the definition in a more mathematically precise way and derive the results just mentioned. We give a derivation of the analogue result of discrete time steps because the derivation is more intuitive.

2.2.2 Definition of Markov Chains

We consider a system characterized by a finite number of states. (We assume that they are numbered 1 to n. At each given moment the system is in exactly one state. The

system transitions randomly to another state with a given, known transition probability. If these probabilities depend only on the state and not on the history of the system, the system is called a Markov chain. If the state transitions happen only at fixed intervals, the Markov chain is called discrete, if they can happen at any time, it is called continuous.

Because we are modeling reliability of a complex system, we can impose a couple of assumptions on a Markov chain.

1. There is exactly one, the absorbing or failure, state from which no transition is possible.

2. All parts of the system are connected to the absorbing state. This means, that we can find a time such that the probability of reaching the absorbing state from any given state in this time is positive.

Because the number of states is finite, the second condition translates into the apparently stronger condition that there exists a time constant T and a lower probability bound P such that the probability of the system reaching the absorbing state from any state in time T is greater than P.

We say that the system has (not) failed if it is (not) in the failure state. We define the reliability vector $\wp(t) =^t (p_1, \ldots p_n)$ whose coefficients $p_i(t)$ is the probability that the system is in the non-failure state i at time t. (The right superscript t refers to the transpose.) The reliability $p(t)$ is the probability that the system has not failed at time t, it is simply the sum of the coefficients of $\wp(t)$.

2.2.3 Discrete Markov Chains

Our goal in this section is to derive an expression for the MTTDL (or more generally, the MTTF) if we assume that transitions take place at the end of a fixed periods. While this is a slightly distorting assumption which yields good results only for small time steps, the derivation is more intuitive and thus more elucidating.

The transition probabilities are collected in the transition matrix M. To be more precise, M is a square matrix whose number of rows is the number of non-failure states. The off-diagonal coefficient $m_{i,j}$ is the probability that the system changes from state j to state i. The diagonal coefficient $m_{i,i}$ is the probability that the system if in state i remains

there and $1 - m_{i,i}$ is the probability that the system leaves state i to go to any other state including the failure state. The reliability vector $\wp$ is given by the difference equation

$$\wp(k+1) = M \cdot \wp(k)$$

and any "initial condition" $\wp(0)$. This implies immediately that

$$\wp(k) = M^t \wp(0).$$

We first derive a techincal lemma that assures us that $\mathcal{I} - M$ with $\mathcal{I}$ the identity matrix is invertible.

Lemma 1 *The geometric series*

$$\sum_{k=0}^{\infty} M^k$$

exists and has value $(\mathcal{I} - M)^{-1}$.

Proof: We know that there is a constant time T and a lower probability bound $P < 1$ such that the probability to go from any state to the failure state in time T is at least P. The probability of non-failure in time T regardless of the initial state is consequentially at most $1 - P$. The probability of non-failure in time $2T$ is at most $(1 - P)(1 - P)$ corresponding to failure during the first T or the second T. By mathematical induction, the probability of non-failure in time lT is then bounded from above by $(1 - P)^l$. In our notation this is:

$$\vec{e} \cdot M^{lT} \cdot \wp(0) \leq (1 - P)^{lT}.$$

(Here $\vec{e}$ is an n-dimensional vector of ones. Multiplication with $\vec{e}$ gives the sum of coefficients of a vector.) For large enough integers s the formula implies:

$$\vec{e} \cdot M^s \cdot \wp(0) \leq a^s$$

with another constant $a < 1$. As all the coefficients of M^s are positive or zero numbers, the equation shows that each coefficient of M^s is smaller than a^s, for s large enough. By comparison with the geometric series $\sum_{k=0}^{\infty} a^k$, the sum

$$\sum_{k=0}^{\infty} M^k$$

exists. If we multiply this sum with $\mathcal{I} - M$, we obtain $\mathcal{I}$. This finishes the proof of the lemma. ■

We are now in a position to propose the main formula of this section.

Theorem 1 $\text{MTTF} = \vec{e} \cdot (\mathcal{I} - M)^{-1} \cdot \wp(0)$.

Proof: As already mentioned, the reliability at time k is given by

$$p(k) = \vec{e} \cdot M^k \cdot \wp(0).$$

The probability for failure at time t is the difference between the failure probability at times k and $k-1$ and given by $(1 - p(k)) - (1 - p(k-1)) = p(k-1) - p(k)$. The MTTF is the expected value for the failure time. We use an index shift to obtain an expression that is useful in itself:

$$\begin{aligned}
\text{MTTF} &= \sum_{k=1}^{\infty} (p(k-1) - p(k)) \\
&= \left(\sum_{k=1}^{\infty} k p(k-1) \right) - \left(\sum_{k=1}^{\infty} k p(k) \right) \\
&= \left(\sum_{k=0}^{\infty} (k+1) p(k) \right) - \left(\sum_{k=1}^{\infty} k p(k) \right) \\
&= \sum_{k=0}^{\infty} p(k)
\end{aligned}$$

We now use the lemma to obtain the desired result:

$$\begin{aligned}
\text{MTTF} &= \sum_{k=0}^{\infty} p(k) \\
&= \sum_{k=0}^{\infty} \vec{e} \cdot M^k \cdot \wp(0) \\
&= \vec{e} \cdot (\mathcal{I} - M)^{-1} \cdot \wp(0).
\end{aligned}$$

The result allows us to reduce MTTDL or MTTF calculation to a matrix inversion.

2.2.4 Continuous Markov Chains

This section presents the analogous result for continuous Markov chains. The transition matrix M is differently defined. The off-diagonal coefficients $m_{i,j}$ of M are the marginal probabilities of state transition from (non-failure) state j to (non-failure) state i and the diagonal coefficients $m_{i,i}$ are the marginal probabilities of leaving state i for either the failure state or another non-failure state. The coefficients of M are thus the transition

rates. The discrete transition matrix corresponds not to our matrix M but rather to $\mathcal{I} + M$. We define the reliability vector $\wp$ and the reliability r as before. From our definitions we have the fundamental differential equation

$$\frac{\delta \wp}{\delta t} = M \cdot \wp.$$

MTTF is the expectation for the failure time t. As failure time is distributed with distribution $1 - p$, MTTF is given by

$$\text{MTTF} = -\int_0^\infty t \frac{dp(t)}{dt} dt.$$

We assume that there exists a time constant T and an upper bound $A < 1$ such that from every initial state we reach the absorbing state in time T with probability at least A. As before we can conclude that the reliability is bound from above by a function $f(t) = a^t$ with constant $a < 1$. This is important to assure convergence in the subsequent calculations.

We can now calculate a more tractable MTTF expression in the following theorem:

Theorem 2 $\text{MTTF} = -\vec{e} M^{-1} \wp(0)$.

Proof: We first use integration by parts to obtain a more accessible MTTF expression:

$$\begin{aligned} \text{MTTF} &= -\int_0^\infty t \frac{dp(t)}{dt} dt \\ &= lim_{t \to \infty}(-tp(t))) \quad + \quad \int_0^\infty p(t) dt \\ &= \int_0^\infty p(t) dt. \end{aligned}$$

The Laplacian of a function $f : \mathcal{R} \to \mathcal{R}$ is defined as $\mathcal{L}(f)(s) = \int_0^\infty f(t) e^{-st} dt$. We note that $\mathcal{L}(f)(0) = \int_0^\infty f(t) dt$. Furthermore, integration by parts yields the differentiation theorem for the Laplacian:

$$\begin{aligned} \mathcal{L}(f')(s) &= \int_0^\infty f'(t) e^{-st} dt \\ &= \left[f(t) e^{-st} \right]_{t=0}^{t=\infty} + s \int_0^\infty f(t) e^{-st} dt \\ &= -f(0) + s\mathcal{L}(f)(s) \end{aligned}$$

We apply the differentiation theorem to the fundamental differential equation:

$$\frac{\delta \wp}{\delta t} = M \cdot \wp,$$

and obtain:

$$\begin{aligned} s\mathcal{L}(\wp)(s) - \wp(0) &= \mathcal{L}(\frac{\delta\wp}{\delta t}) \\ &= M \cdot \mathcal{L}\wp(s). \end{aligned}$$

If we now set s equal to zero, we have:

$$-\wp(0) = M \cdot \mathcal{L}(\wp)(0) = \int_0^\infty \wp(t)dt$$

We now put everything together to obtain the theorem:

$$\begin{aligned} \text{MTTF} &= \int_0^\infty p(t)dt \\ &= \vec{e} \cdot \int_0^\infty \wp(t)dt \\ &= \vec{e} \cdot \mathcal{L}(\wp)(0) \\ &= -\vec{e} \cdot M^{-1}\wp(0). \end{aligned}$$

■

The theorem reduces MTTDL calculations to the determination of the transition matrix M and the calculation of its inverse.

2.2.5 Numerical Aspects

There are several good algorithms known to take the inverse of a large matrix. The first procedure is the method of Gauss Jordan. The basic idea is simple: The matrix is positioned next to the identity matrix and then elementary row transformations will transform the matrix into the identity matrix, while the same transformations will transform the identity matrix into the inverse matrix. More concretely, the method proceeds as follows: Starting with column one, a pivot element is chosen and the corresponding row swapped with the first row, if necessary. Then the first row is divided by the pivot. Then we subtract a multiple of the first row from all other rows such that all other elements in the first column become zero. Then we proceed similarly with the second column. We pick a pivot not in the first row, switch the row in which the pivot is located with the second row and proceed to use row-subtraction to render all other elements in the second column zero. We proceed until the matrix has been transformed into the identity matrix. The same transformations applied to the identity matrix yield the inverse matrix.

The crux of the method is the choice of the most desirable pivot. Simply choosing the (by magnitude) largest available pivot is a good method. We use implicit pivoting, which chooses the largest available pivot if the largest elements in all rows would have been the same. Fortunately, all our transition matrices tend to have the largest coefficients located on the main diagonal and the choice of pivots will follow it. Full pivoting will pick the best pivot in any column not touched upon so far and interchange columns. The column interchange scrambles the order of rows in the inverse matrix. Because of the special nature of the matrices, we can limit ourselves to partial pivoting, that is without column interchanges.

Gauss Jordan method tends to be unstable numerically without pivoting, is somewhat slow and in the form presented is storage intensive. However, we can superimpose both matrices, the one formed by row-transformations from the original matrix and the one formed form the identity matrix and thus save space.

The second method uses LU decomposition of the matrix. The matrix is written as a product of a lower and an upper diagonal matrix. Both factors can be inverted easily and the product (in different order) is the inverse.

To determine the impact of numerical errors on the calculation of MTTF we multiply the numerical inverse $\tilde{M}$ of M with M and write the result as

$$M \cdot \tilde{M} = \mathcal{I} + \Delta$$

with error matrix Δ. The size of the largest coefficient of Δ is 10^{δ}. The size of the largest coefficient of M^{-1} is 10^{m}. Then

$$-\vec{e}\tilde{M}\wp(0) = -\vec{e}M^{-1}M\tilde{M}\wp(0) = -\vec{e}M^{-1}\wp(0) + -\vec{e}M^{-1}\Delta\wp(0)$$

gives us an estimate of the error by replacing the unknown quantity M^{-1} with the known $\tilde{M}$. Because $\wp(0)$ is a unit vector, the error is at most of size $n^3 \cdot 10^{m+\delta}$. Most of the actual Markov models have small to moderate dimensions n and the error is not noticeable. Even in our biggest model the first 3 decimal digits are valid. As the approximation of the error is easy to calculate, we use it in our calculations to flag untrustworthy results.

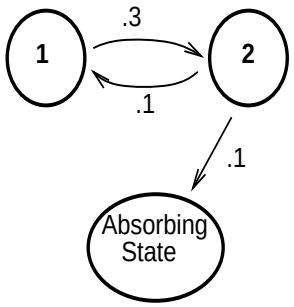


Figure 2.2: An Simple Example for a Markov Chain

2.2.6 An Example

We include a simple example of a Markov model to help the reader familiarize oneself with our calculational method of computing Mean Time to Failure. In Figure 2.2 we present three state Markov Model.

Assume that the starting state is 1. Let p_1 and p_2 denote the probabilities that the system is in State 1 or State 2 respectively. Let us first assume that our Model is discrete, that is, that at fixed times the state changes according to the given probabilities. Then the new probabilities can be calculated from the old ones by

$$\begin{aligned} p_1(t+1) &= .7p_1(t) + .1p_2(t) \\ p_2(t+1) &= .3p_1(t) + .8p_2(t). \end{aligned}$$

or by

$$\begin{pmatrix} p_1(t+1) \\ p_2(t+1) \end{pmatrix} = \begin{pmatrix} .7 & .1 \\ .3 & .8 \end{pmatrix} \cdot \begin{pmatrix} p_1(t) \\ p_2(t) \end{pmatrix}.$$

Then

$$(\mathcal{I} - M)^{-1} = \begin{pmatrix} 6.67 & 3.33 \\ 10 & 10 \end{pmatrix}$$

and we obtain 16.67 for the MTTF from starting state 1, namely the sum of coefficients of the first column. This number has as dimension the number of possible state transaction.

We now assume that the Markov model is continuous. Then

$$\begin{aligned} \frac{\delta p_1}{\delta t} &= -.3p_1(t) + .1p_2(t) \\ \frac{\delta p_2}{\delta t} &= .3p_1(t) - .2p_2(t). \end{aligned}$$

Hence, the transition matrix M is almost the same, differing by the addend $\mathcal{I}$.

$$M = \begin{pmatrix} -.3 & .1 \\ .3 & -.2 \end{pmatrix}$$

We can interpret the coefficients immediately from the diagram. A transition from State i to State j is represented in the transition matrix by lowering the (i,i) entry by the transition probability and making the transition probability the (j,i) entry.

2.3 Disk Array Components and Failure Rates

We consider three varieties of components within a disk array subsystem. There are (i) individual disk drives, (ii) strings of disks and (iii) essential components which, by definition are common to all disks such that failure leads to complete data inavailability. We do not model operation failure in this work though it can have a significant effect on reliability. However, our current results will point in the proper direction when beginning such an effort. For our reliability calculations, we assume that components fail with exponential (memory-less) probability. While this assumption ignores the well-known "bathtub" life expectancy of components it yields both excellent approximations and bases for a design decision.

The most crucial factor in disk array reliability and costs is the frequency of repair. In the extreme, a response time of a few minutes is both expensive and phenomenally efficient. We assume that a typical response time is 35 hours, which corresponds to a repair the next work day.

The basic components of the disk array are, of course, the individual disk drives. The life expectancy of disk drives has increased considerably over the last decade. For example, Schulte [32] and Gibson [11] in the late eighties use a MTTF of 50,000 hours. Today (1993) MTTF quotes of 500,000 hours are common. In the past producers have tended to understate the MTTF of their products. This policy counteracts a common misunderstanding that MTTF is a guaranteed life expectancy instead of average life expectancy. For example, if the failure probability is memory-less, three failures are observed during a single MTTF interval in about 8% of all cases. It is notoriously difficult to predict future technological development and we refrain from speculation on the longevity of future disks. As disks contain mechanical parts, we can expect disk MTTF to be small compared with purely electronic devices.

A disk array needs certain central elements, a disk array controller, a non-volatile cache to store data to be stored to the disk array and intermediate results like old information and check data, and an encoder. These elements are crucial, their failure disables the array. As they are electronic, their reliance is very high.

A string is a collection of disk drives sharing several hardware components: power

event		rates (per hour)	identifier
essential component failure		1×10^{-7}	ϵ
disk drive failure		2×10^{-5}	λ
string failure	soft	2×10^{-5}	μ
	hardened	5×10^{-6}	
	super-hardened	5×10^{-8}	
component repair		2.77×10^{-2}	ρ

Table 2.1: Event Rates

supply, cooling, cabling, host bus adapters (HBA) and small computer system interface (SCSI) controllers. An HBA can serve up to five disks, if our strings contain more disks, than there are the appropriate number of HBAs. We consider varieties of hardware redundancy within strings: A string is *soft* if it contains only the basic set of hardware components. A string with ten disks would have two sets of HBAs and SCSI controllers, one power supply and one cooling fan. The string MTTF for such an ensemble is around 50,000 hours. A string is *hardened* if some components are duplicated. Within this report our hardened strings will double the power supply, SCSI controllers and HBAs but only contain one fan. The MTTF is increased to about 200,000 hours. A *superhardened* string doubles the cooling and might even provide additional HBA and SCSI controller redundancy. This configuration has a MTTF of more than 1,000,000 hours.

With a certain probability, one of the essential disk array components will fail. We assume that the disk array is hardened against power failure, otherwise, we are at the mercy of the power company and MTTF are low.

In Table 2.1 we summarize transition event rates used in our calculations.

Chapter 3

Level 5 RAIDs

3.1 Overview

Disk accesses are inherently slower than main memory accesses by a factor of about 10^5. (In 1993, a DRAM access is typically between 70 and 120 nanoseconds, whereas a disk access takes about 5 to 40 milliseconds.) The discrepancy between secondary memory and processor speeds makes highly parallel storage elements like a farm of independent disks very enticing. However, the MTTF of a system of n independent, but crucial, components is $1/n$ of the MTTF of a single component, so that the reliability of such a disk farm is bad. For example, a disk farm with 100 disks would have a MTTDL of 21 to 210 days only. Mirrored disks and mirrored systems have been used for quite a while in failure tolerant systems. While the reliability is very attractive, the hardware costs are not. A less expensive solution was presented by Patterson, Gibson, Karp, Katz, Hellerstein [28], [11], [12]. By introducing redundancy in a less drastic way performance, reliability and costs become acceptable.

The literature distinguishes five varieties of RAID organizations. In this thesis, we are only interested in the Level 5 RAID. The disks in a Level 5 RAID are divided in mutually exclusive reliability groups, each containing one extra "check" disk which stores the parity of the other disks in the reliability group. As the parity together with all but one disk contents yields the remaining disk contents, the ensemble tolerates failure of one disk without loosing access to any data. Under fault-free operating conditions, a read operation proceeds by reading directly from the data storing "information" or "message" disk. A

write is more complicated, as we need to update parity as well. Instead of recalculating the parity data, which would involve reading the other disks in the reliability group, we calculate the partity as the parity of the new and old information and the old parity data. (If we wrote to an empty sector, then the old information is assumed to be a string of zero bits.) Every write at the "logical" level involves two read operations and two write operation as well as some arithmetic. During a write burst a disk, that would exclusively store parity, would be quite busy and become a performance bottleneck. For that reason the Level 5 RAID organization disperses the parity information throughout the reliability group. This is an instance of an addressing scheme for load-balancing: For example, the parity for the first track might be stored on the first disk, the parity for the second track on the second disk and so on. We can implement this by using a circular shifting permutation for disk addresses, that would map logical disk 1 (the check disk) on physical disk 1 for track number 1, logical disk 1 on physical disk 2 for track number 2 and so on.

The complexity of the write operation introduces a time window of vulnerability into the RAID: If a power failure disables the RAID momentarily during an update and another disk in the reliability group fails, or if the disks to which we write is disabled before the parity can be updated, then the check data do not reflect the parity and the disk failure leads to data-loss. If we use disk write synchronization, where we try to update both information and check disks as simultaneously as possible, then the performance deteriorates intolerably. We simply introduce a non-volatile cache into the array organization to solve the problem. The cache preserves the data of all write operations to which the RAID has committed even through the event of a malfunction and allows the RAID to restart operations interrupted by such an event. Hence, the parity information always reflects the data on the message disks in a reliability group.

To minimize the impact of string failure, Gibson [11], see also [32], introduces an orthogonal arrangement of strings and reliability groups. Each string contains exactly one disk in each reliability group and vice versa.

Introduction of spare disks increases considerably the reliability. If a disk fails, its contents - both information and check data - are reconstructed on the spare disk. The spare disk serves such as a *pro tempore* self-repair tool. The reconstruction time can be as short as a few minutes (with noticeable performance degradation) or long enough to render

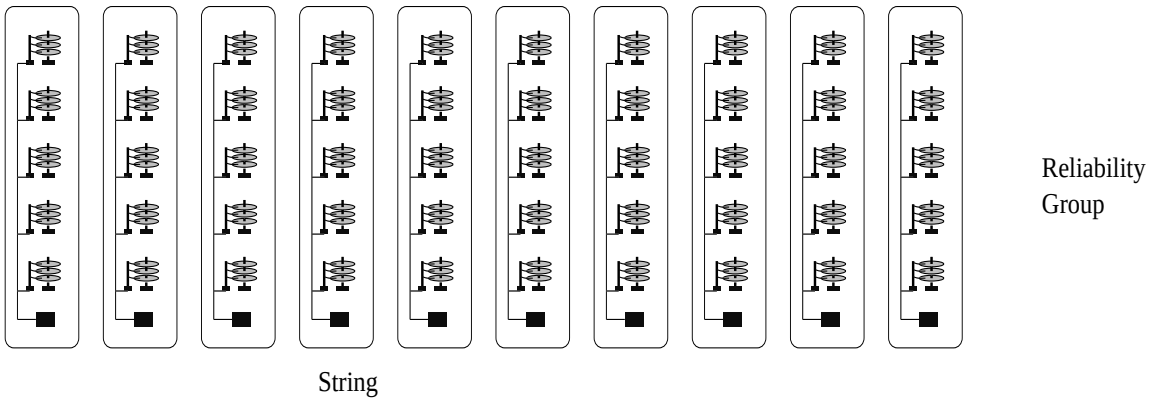


Figure 3.1: Disk Array Ensemble with Ten Strings and Five Reliability Groups

the performance degradation negligible. We can use on-demand reconstruction, which only reconstructs data that are being read, forced reconstruction, which rebuilds the contents of the failed disk independently of demand, and any combination between these two principal possibilities.

3.2 Level 5 RAID Organizations and Reliability Results

Inside the Level 5 Raid organization we distinguish a number of subschemes, whose reliability we investigate in the following sections. The distinguishing elements between the different schemes are the number and arrangement of spares and the extent of load balancing through disk address translation.

We fix notations: m denotes the number of reliability groups, n the number of disks in a reliability group including the check disks and the spare disks, if present.

We determine the reliability figures for the different RAIDs organizations for a basic RAID with 5 disks per string and 10 strings for a storage capacity of mostly 45 disks as depicted in Figure 3.1. We discuss some of the Markov models for an additional example of a RAID with 10 disks per string and 11 strings, to reassure the reader and us that the Markov models are applicable over a wide range of RAID dimensions.

3.2.1 Declustering Level 5 RAIDs

We call a RAID which has the orthogonal arrangement of reliability groups and strings (as in Figure 3.1) a *classic* RAID. If the classic Level 5 RAID suffers a disk failure, the load and the utilization of the disks in the same reliability group almost doubles. We can avoid the resulting loss of performance by spreading the additional work over all the other disks in the RAID. The literature calls this process *clustering* [25] or *declustering* [14]. Because (de)clustering equalizes the disk loads not only under fault-free conditions, we refer to it as a form of load balancing and distinguish complete and almost complete load balancing. In Section 3.3.3 we introduce a new method for load balancing, the Almost Complete Address Translation Scheme (ACATS). ACATS introduces a layer of virtual disks that are organized in the same orthogonal arrangement as a classic Level 5 RAID. By permuting separately the string and the disk-in-string addresses, we provide address translation to the actual disks. In contrast to the load balancing scheme given by Holland in [14] our scheme respects track and block addresses. This allows for easier reconstruction on spares.

3.2.2 Level 5 Raids with Hot Stand-By Disks

A dramatic reliability enhancement for disk arrays is obtained by shortening the repair times. As we assume that components fail independently, two failures in a short repair interval become very unlikely and with smaller repair times, the chance of data loss becomes arbitrarily small. We can simulate these short repair times by providing spare components, that can take over until we can replace the faulty device.

We investigate the provision of a small number of spare disks and the provision of a full spare string. Spares provide excellent reliability against disk and, in the latter case, string failure. Our results are based on the assumption of independent component failure and have to be read with this *caveat* in mind.

We can provide stand-by protection by (a) providing spare disks on their own string or (b) by providing spare space throughout the disk array. The latter is called *distributed sparing* and was introduced by [22]. Distributed sparing uses the spare disks under fault-free conditions and has then better performance.

In the event of disk failure, the contents of a failed disk are *reconstructed* on spare space. We can reconstruct a now inaccessible data block *on demand,* that is, when the

block is read and needs to be reconstructed anyway, or independently. In practice, some compromise between these two strategies could be pursued. The total time to reconstruction can become an important reliability factor, if the time is large. We will assume that it is short enough to be indistinguishable from being instantaneous.

We use distributed sparing to provide as little as one disk's worth of spare space. We distinguish two subschemes, called *Naive Distributed Sparing* (NDS) and *Safe Distributed Sparing* (SDS). The former scheme just reconstructs the data on the failed disk, whereas the second scheme treats the reconstruction writes as RAID writes. We will see that SDS gives indeed better reliability over NDS and that the performance in the single disk failure case is not significantly worse than NDS (see Section 11.2.5.)

3.2.3 Comparison of RAID MTTDL for RAIDS with Different Storage Capacities

The MTTDL of some of our schemes are hard to compare, as the storage capacities differ slightly. We propose here the *"Single Disk Equivalent MTTDL"* (SDE MTTDL) as an measure. We assume that the same amount of data is stored on a set of disks of the same size. We then calculate the MTTF that these disks would have to evidence to achieve the same MTTDL as the RAID organization. As the MTTF of an ensemble of N components is $1/N$ times the MTTF of a single component, we obtain the SDE MTTDL by multiplying the MTTDL of a RAID organization by the storage capacity measured in numbers of disks. The higher the SDE measure, the higher is the reliability. We give the SDE MTTDL for organizations with one or two spares in Table 3.2.

In Section 3.3 we give an overview over all Level 5 RAID organizations that we analyze. Section 3.4 then presents in the same order the derivations and the details of the Markov models. The subsection numbers in Sections 3.3 and 3.4 correspond directly. We conclude Section 3.3 by presenting Tables 3.1 and 3.2 which present sample mean time to data loss results. As always, we use the parameters specified in Table 2.1.

3.3 Overview of Level 5 RAID Organizations

3.3.1 Classic Level 5 RAID

Our orthogonal arrangement of reliability groups and strings is the same as that in Gibson [11]. (See Figure 3.1.)

Our first organization does not use spares and uses a simple address translation scheme, in which disk addresses only inside a reliability group are permuted. This is the set-up that has been investigated by Schulze [32] and Gibson [11]. Data loss occurs if two disks in a reliability group are inaccessible.

3.3.2 Level 5 RAID with Complete Address Translation

If the classic Level 5 RAID suffers a disk failure, the load and the utilization of the disks in the same reliability group almost doubles. (See Chapter 10.) We can avoid the resulting loss of performance by spreading the additional work over all the other disks in the RAID.

We implement complete load balancing with the least discriminate track address translation: We permute all the disks in the disk array. To be more precise, if we write to the logical disk address (D, τ) where D is the number of a disk and τ is a track (or even block) number, then the physical address is $(\sigma_\tau(D), \tau)$ Here, σ denotes a disk address permutation, which depends on the track number τ. If the permutation is chosen quasi-randomly, we achieve almost complete load balancing. If a disk failed, then every disk has some tracks which belong to the same reliability group as tracks on the lost disk. Consequentially, every disk has a fair share in the reconstruction process for lost data. (We talk about the implementations of such an addressing scheme in more detail in Chapter 6.) Our Complete Address Translation Scheme (CATS) is an instance of declustering.

The RAID will suffer data loss if two tracks belonging to the same reliability group are inaccessible. This happens when two disks or one string fail. The Markov model (Figure 3.6 has only two states and we can give MTTF in a closed, though not very elucidating form in Theorem 3.

Theorem 3 *The MTTDL for a Level 5 RAID with complete load balancing for a failed*

disk is given by

$$\text{MTTDL} = \frac{\rho + \epsilon + (2nm-1)\lambda + n\mu}{(\epsilon + nm\lambda + n\mu)(\epsilon + (nm-1)\lambda + n\mu + \rho) - \rho nm\lambda}$$

3.3.3 Level 5 RAID with Almost Complete Disk Address Translation (ACATS)

The preceding organization was vulnerable to single string failure. With only a small change in the address translation scheme we can achieve almost the same performance benefits in the disk failure case while gaining tolerance against single disk failure.

The almost complete disk address translation scheme (ACATS) never places tracks on disks in the same string in the same reliability group. To implement it, we address disks with a string number and a disk-in-string address. We permute string addresses and then disk-in-string addresses. That is, we first pick a string and then a disk on that string. A track τ on the (logical) disk located in string S_i in the j^{th} position on the string is then located on the physical disk on string $S_{\sigma(i)}$ and with disk-in-string address $\tilde{\sigma}(j)$.

The disk array can now tolerate both a disk failure and a string failure but not both. In some instances failure of two or even more disks does not lead to data loss, namely if the failed disks are all located on the same string.

As the Markov model (depicted in Figure 3.6) has only two non-failure states, we can give the MTTDL result in closed form.

Theorem 4 *The MTTDL for a Level 5 RAID with almost complete load balancing for a failed disk is given by*

$$\text{MTTDL} = \frac{\rho + \epsilon + (2n-1)m\lambda + (2n-1)\mu}{(\epsilon + nm\lambda + n\mu)(\rho + \epsilon + (n-1)m\lambda + (n-1)\mu) - \rho(nm\lambda + n\mu)}$$

A RAID with CATS is only two disk failures or one string failure away from data loss. This extreme vulnerability is expressed in the low MTTDL values in Table 3.1. A RAID with ACATS can withstand two string failures, but still only two disk failures in different strings. In contrast, the classic Level 5 RAID can withstand most failures of 2 disks, which leads to the higher reliability values. We can see the higher reliability of the classic scheme over ACATS, or any other declustering scheme, for all values in Table 3.1. In the following schemes we introduce spare space or spare disks in the classic and the

ACATS RAID organizations. As the tolerance against disk failures improves, the better disk failure tolerance of the classic scheme becomes less important and the difference in reliability between the two schemes is diminuished, only to widen again somewhat, when we introduce a full spare string into the two organizations and provide the capability of withstanding two string failures.

3.3.4 Classic Level 5 RAID with Distributed Disk Sparing

Our first disk array organization with spares is built on top of the classic Level 5 RAID. The equivalent of one or two of the nm disks in the RAID is set apart as a spare, but with distributed sparing one track among all nm tracks with the same track address is used as a spare track. Then we distribute the spare track evenly throughout the disk array. This organization combines good performance with good reliability against disk failure, but does not increase resilience against string failure. According to the nature of the update we distinguish NDS (Naive Distributed Sparing) and SDS (Safe Distributed Sparing.) The latter in connection with super hardened strings already achieves fully satisfactory reliability.

3.3.5 Level 5 RAID with Distributed Disk Sparing and Almost Complete Load Balancing

Our second organization with spares uses ACATS, which we introduced in 3.3.3. We reserve one or two logical information disk as spare space. The address translation scheme assures that all physical disks will carry the spare space in almost equal proportions. The contents of the first failed disk, whether information or check, will be reconstructed on the spare spaces. Only one disk will contain a spare track for any given track number.

While slightly worse than in the preceding scheme, SDS in conjunction with super hardened strings achieves impressive reliability. As the performance differences between SDS and NDS are not substantial, the higher reliability numbers indicate use of SDS.

3.3.6 Classic Level 5 RAID with a String of Spares

In contrast to distributed disk sparing, the present scheme uses dedicated spares located on a special smaller string, the string of spares. This scheme achieves comparable reliability with SDS distributed sparing. While it offers a simpler organization, the scheme

does not use the spare disks to lower utilization in the non-failure state and has one more string. We assume that the string of spares is smaller than a normal string.

3.3.7 Level 5 RAID with a Distributed String of Spares and Almost Complete Load Balancing

This organization is the companion scheme to the previous scheme (Section 3.3.6 and provides a small set of spares on a string. The reliability is somewhat smaller than for the Classic RAID, but the considerable disk failure tolerance is shortening the gap.

The load balancing addressing scheme is the same as in Section 3.3.3. As a consequence, the effects of two disk failures in the same string are less severe than two disk failures in different strings. Our scheme uses reassignment of spares. The gains of this procedure are limited, as the occasion only arises infrequently.

3.3.8 Classic Level 5 RAID with a Distributed Spare String

We now consider organizations which are capable of withstanding even two string failures without data loss. Our first organization, by [22] provides a full spare string, but keeps the classic fixed assignment of reliability groups. We distribute the spare string just like the string of checks. In case of disk failure, the reliability group uses the spare space to reconstruct lost message or check data. If a second disk fails, available spare space from another reliability group is "borrowed." An egoistic strategy that reserves spare space for the reliability group it is located in leads to the same reliability as the MDS based RAIDS considered later. We reassign spare space used by one reliability group under certain circumstances. This procedure, which can be executed quite leisurely, achieves another, smaller reliability boost. This scheme shows the best reliability of all schemes considered here.

3.3.9 Distributed String Sparing with Almost Complete Load Balancing

The second organization combines Distributed Sparing with ACATS. An ACATS uses string permutations as one of the two steps of address translations and hence blends very well with distributed sparing. The resulting reliability falls short of the Classic scheme but is still very impressive.

Disk Array Organization	Storage Capacity (disks)	MTTDL (in years) Super Strings	MTTDL (in years) Hard Strings	MTTDL (in years) Soft Strings
No Redundancy	45	0.11	0.11	0.10
3.3.1: Classic	45	17.40	11.84	5.76
(3.3.2: CATS	45	3.46	1.41	0.50
(3.3.3): ACATS	45	3.73	3.40	2.63
(3.3.4): Classic (NDS) 1 Spare	44	162.64	29.93	8.11
Classic (NDS) 2 Spares	43	640.32	34.66	8.41
Classic (SDS) 1 Spare	44	335.56	56.75	14.33
Classic (SDS) 2 Spares	44	862.91	65.04	15.09
(3.3.5): ACATS (NDS) 1 Spare	44	91.46	26.20	7.81
ACATS (NDS) 2 Spares	44	613.87	34.58	8.41
ACATS (SDS) 1 Spare	44	98.21	40.36	13.02
ACATS (SDS) 2 Spares	43	719.23	64.08	15.04
(3.3.6): Classic 1 Sp. Str.	45	326.95	65.61	16.60
Classic 2 Sp. Str.	45	885.18	78.22	17.82
(3.3.7): ACATS 1 Sp. Str.	45	94.64	38.98	12.50
ACATS 2 Sp. Str.	45	683.70	58.29	13.64
(3.3.8): Classic Distr. String Sparing	45	1133.85	776.74	286.01
(3.3.9): ACATS Distr. String Sparing	45	1119.00	496.84	157.88

Table 3.1: MTTDL Values for Level 5 RAID Organizations

Disk Array Organization	SDE MTTDL (in years) Super Strings	SDE MTTDL (in years) Hard Strings	SDE MTTDL (in years) Soft Strings
Classic (NDS) 1 Sp.	7156	1317	357
Classic (NDS) 2 Sp.	27534	1490	362
Classic (SDS) 1 Sp.	14765	2497	631
Classic (SDS) 2 Sp.	37105	2797	649
ACATS (NDS) 1 Sp.	4024	1153	344
ACATS (NDS) 2 Sp.	26396	1487	349
ACATS (SDS) 1 Sp.	4321	1776	573
ACATS (SDS) 2 Sp.	30927	2755	647
Classic Sp. Str. (1)	14713	2952	747
Classic Sp. Str. (2)	39833	3520	802
ACATS Sp. Str. (1)	4259	1754	563
ACATS Sp. Str. (2)	30767	2623	614

Table 3.2: Single Disk Equivalent MTTDL Values

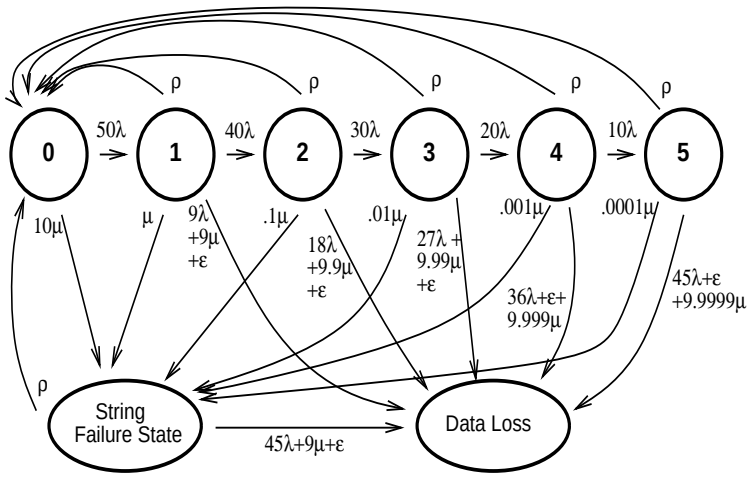


Figure 3.2: Markov Model for the Classic Raid Organization with 5 Reliability Groups and 10 Strings.

3.4 Modeling Reliability

The states of our Markov models describe the failure of components and their repair. In all our models, a transition describing repair will go from all non-failure states to the fault-free state. A transition with probability ϵ describes the failure of an essential component. We assume ϵ to be quite low, as the essential components are all non-mechanical devices.

3.4.1 Classic Level 5 RAID

In our first model, the disk array contains no spare disks and address translation is only done inside a reliability group. The array suffers data loss only if two or more disks in the same reliability group become inaccessible. We call a reliability group with one failed disk *vulnerable*. The number of vulnerable groups label the non-failure states of our Markov model, which we give in Figure 3.2. State 0 corresponds to the fault-free state in which every RAID component is functioning. Additionally, there is a string failure state. There is a subtle, but important difference between state n and the string failure state. While the number of failed disks in both states is the same, in the latter case, they are all located on

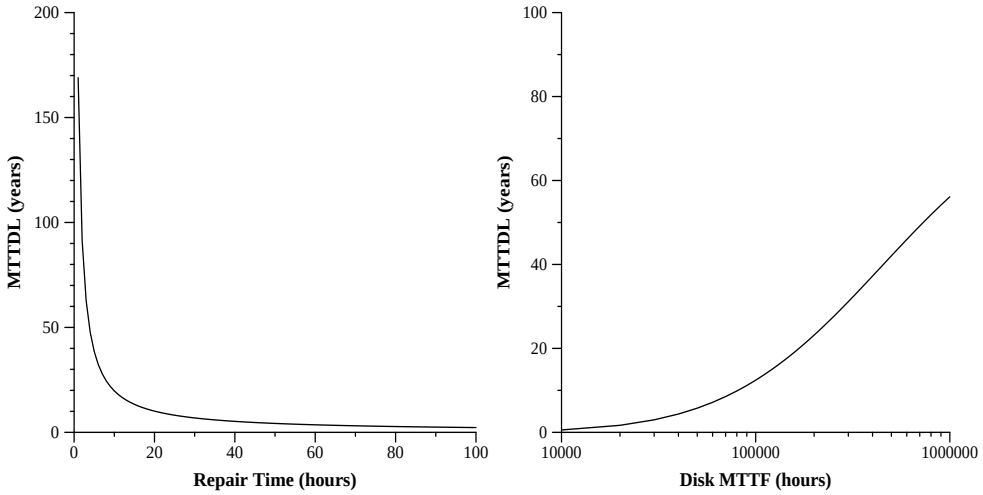


Figure 3.3: Sensitivity Analysis for the Classic Level 5 RAID with soft strings: MTTDL Dependency on the Repair Time (left) and on Disk MTTF (right).

the same string, which is not necessarily true in the former state. The probabilities of data loss generating string failure are different, which is reflected in different transition rates to the failure state.

Repair transitions from every non-failure state with the exception of the fault-free state to the fault-free state with marginal probability ρ. Essential component failure transitions from any non-failure state to the failure state with marginal probability ϵ.

Disk failure in a non-vulnerable reliability group results in a transition to the next higher state and disk failure in a vulnerable group to data loss. From state i the former transition is taken with marginal probability $(m-i)n\lambda$ and the latter with probability $in\lambda$. A string failure either leads to transition to the string failure state (where all groups are vulnerable) or to a transition to the data loss state. The former transition happens if all previously failed disks are located on the failing string, so that the marginal probability for the transition from state i to state m amounts to $(1/n^i)n\mu$. The string failure component of the marginal probability to transition from state i to the failure state is $(1-1/n^i)n\mu$. The RAID has lost all redundancy in the string failure state and suffers data loss if any components fail.

We present the Markov model in Figure 3.2. The result of our sample calculations

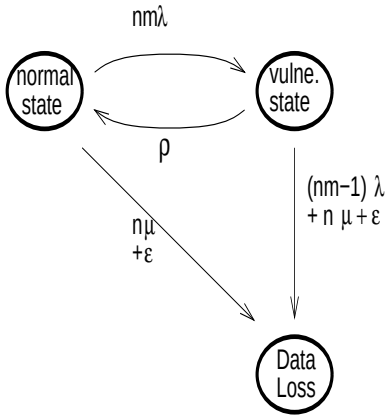


Figure 3.4: Markov Model for the Level 5 RAID with Complete Disk Address Translation

are shown in Table 3.1. In Figure 3.3 we give a small study on the influence of repair time and disk MTTF. We see that very short repair times boost the MTTDL rating. The parameters are given in Table 2.1.

3.4.2 Level 5 RAID with Complete Address Translation

The classic Level 5 RAID with Complete Address Translation (CATS) places a given track of a given disk (seemingly) randomly in a given reliability group. As a result of this shuffling, the RAID loses data with extremely high probability if any two disks fail.

Let us prove this statement. Two randomly chosen disks are located in the same reliability group with probability $p = (n-1)/(nm-1)$. With complete load balancing, there is however not one but 1000 chances (the number of tracks) for two given physical disks to be contained in the same reliability group for at least one track number. The probability for data loss is then $1-(1-p)^{1000}$ which is close enough to one for all practical purposes. (For example for the small RAID with 5*9 disks we obtain 4×10^{-42} for the data survival chances with two failed disks.) Our chances to avoid data loss are even worse if a whole string has failed. In our calculation we assumed that CATS assigns disks to reliability groups randomly. In reality, CATS uses a pseudo-random process to generate a fixed addressing scheme. If the generation process is not very good, it is possible that certain pairs of disks will never be assigned to the same reliability group. Perversely, this

increases reliability, but not by much. We will always assume in reliability calculations that the addressing schemes obtained through pseudo-random assignments of disks are indeed random.

Our Markov model (in Figure 3.4) has only two non-failure states: the fault-free state and a state indicating loss of a single disk. A repair corresponds to a state transition with probability ρ from the failed disk state to the fault-free state. An essential component failure is indicated by a transition with probability ϵ from both non-failure states to the failure state. A disk failure moves the system with probability $nm\lambda$ from the fault-free state to the failed disk state. A transition from both non-failure states to the failure state with probability $n\mu$ describes a string failure. A second disk failure is represented by a transition from the failed disk state to the failure state, which is taken with probability $(nm-1)\lambda$. We summarize the transitions in the transition matrix M.

$$M = \begin{pmatrix} -\epsilon - nm\lambda - n\mu & \rho \\ nm\lambda & -\rho - \epsilon - n\mu - (nm-1)\lambda \end{pmatrix}.$$

The determinant of M is

$$det(M) = (\epsilon + nm\lambda + n\mu)(\rho + \epsilon + n\mu + (nm-1)\lambda - \rho nm\lambda$$

and the inverse of M is

$$M^{-1} = det(M)^{-1} \cdot \begin{pmatrix} -\rho - \epsilon - n\mu - (nm-1)\lambda & -\rho \\ -nm\lambda & -\epsilon - nm\lambda - n\mu \end{pmatrix}$$

and we obtain the MTTF as the negative sum of the first column. This proves theorem 3.

We use the RAID to again explore the sensitivities to important parameters. In Figure 3.5 we present the RAID MTTDL dependency on the repair and on the Disk MTTF. The parameters are again the ones in Table 2.1. We can see that shortening the repair time can have dramatic effects and that the efficacy of lengthening the Disk MTTF looses beyond the point when other failure causes become dominant.

3.4.3 The Level 5 RAID with Almost Complete Load Balancing

Almost Complete Load Balancing is implemented by the Almost Complete Address Translation Scheme (ACATS). This scheme never places tracks of the same reliability group

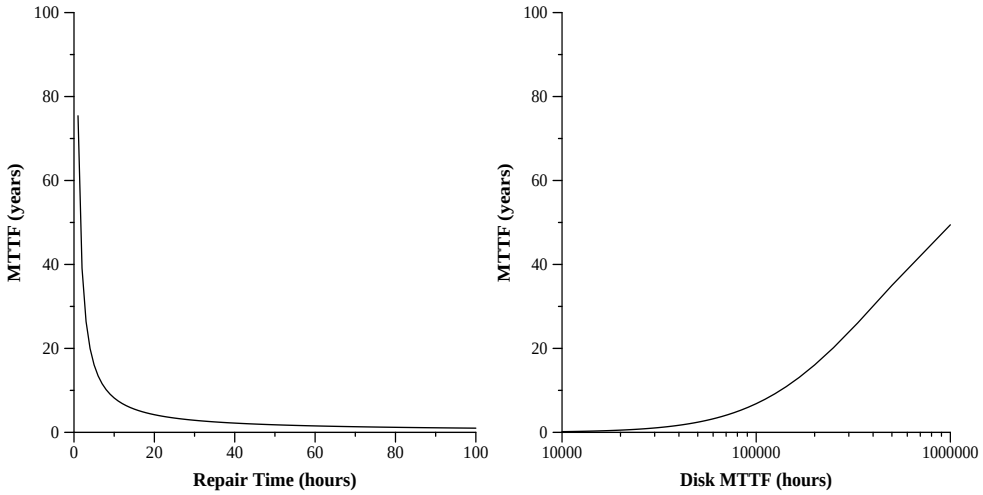


Figure 3.5: Sensitivity Analysis for the CATS RAID with soft strings: MTTDL Dependency on the Repair Time (left) and on Disk MTTF (right).

on the same string. However, similarly as in 3.4.2, the probability that the inaccessibility of two disks on different strings leads to data loss is almost undistinguishable from 1. We can determine the MTTDL from the Markov model in Figure 3.6 with only two non-failure states. The first one describes the fault-free operating mode, the second depicts a *vulnerable* situation in which one string or disks on one string have failed.

As usual, we have a repair transition from the vulnerable state to the fault-free state as well as an essential component failure transitioning from either state to the failure state. In the fault-free state, a transition to the vulnerable state takes place with probability $nm\lambda+n\mu$, corresponding to disk and string failure. In the vulnerable state, only disk failures outside the already afflicted string matter, hence we observe a transition to the failure state with probability $(n-1)m\lambda+(n-1)\mu$. We obtain for the transition matrix M:

$$M = \begin{pmatrix} -\epsilon - nm\lambda - n\mu & \rho \\ nm\lambda + n\mu & -\rho - \epsilon - (n-1)\mu - (n-1)m\lambda \end{pmatrix}.$$

The determinant of M is

$$det(M) = (\epsilon + nm\lambda + n\mu)(\rho + \epsilon + (n-1)m\lambda + (n-1)\mu) - \rho(nm\lambda + n\mu)$$

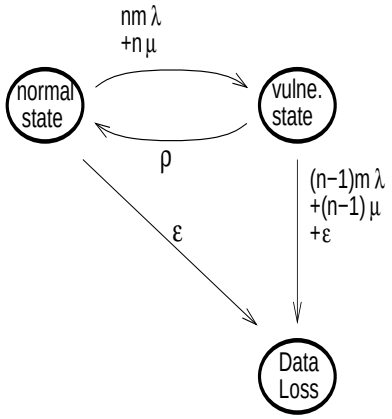


Figure 3.6: Markov Model for the Level 5 RAID with ACATS

and the inverse of M is

$$M^{-1} = det(M)^{-1} \cdot \begin{pmatrix} -\rho - \epsilon - (n-1)\mu - (n-1)m\lambda & -\rho \\ -nm\lambda - n\mu & -\epsilon - nm\lambda - n\mu \end{pmatrix}$$

MTTDL is now given by the negative sum of the first column of the inverse matrix:

$$\text{MTTDL} = \frac{\rho + \epsilon + (2n-1)m\lambda + (2n-1)\mu}{(\epsilon + nm\lambda + n\mu)(\rho + \epsilon + (n-1)m\lambda + (n-1)\mu) - \rho(nm\lambda + n\mu)}$$

which proves Theorem 4.

(a)

track 1:		i_1	i_1	c_1
	i_2	i_2	i_2	c_2
track 2:	c_1	i_1	i_1	i_1
		i_2	i_2	c_2
track 3:	c_1		i_1	c_1
	c_2	i_2	i_2	i_2
track 4:	c_1	i_1	i_1	i_1
	c_2		i_2	i_2
track 5:	i_1	c_1		i_1
	i_2	i_2	i_2	c_2
track 6:	c_1	i_1	i_1	i_1
	i_2	c_2		i_2

(b)

track 1:	c_1	i_1	i_1	■
	i_2	i_2	i_2	c_2
track 2:	c_1	i_1	i_1	■
	i_1	i_2	i_2	c_2
track 3:	c_1	i_1	i_1	■
	c_2	i_2	i_2	i_2
track 4:	c_1	i_1	i_1	■
	c_2	i_1	i_2	i_2
track 5:	i_1	c_1	i_1	■
	i_2	i_2	i_2	c_2
track 6:	c_1	i_1	i_1	■
	i_2	c_2	i_1	i_2

Figure 3.7: A Two By Four Disk Array: (a) Fault-Free Data Lay-Out (b) Data Lay-Out After Disk Failure

3.4.4 Classic Level 5 RAID with Distributed Disk Sparing

Our organization introduces spare space into the classic Level 5 RAID organization. At the track level, a single track in the whole RAID is reserved as spare space. This decreases the storage capacity to a very tolerable $(nm-1)/nm$ of the total capacity. We first present the *Naive Distributed Sparing* (NDS) scheme. When a disk fails, its contents are reconfigured and placed on the spare space. In contrast, *Safe Distributed Sparing* (SDS) updates the check data of the reliability group, in which the spare track happens to be situated as well. Adding one further spare track increases reliability consistently and we give the reliability numbers, too.

This organization can tolerate any two disk failures or a string failure without data loss. However, a disk failure followed by a string failure can lead to data loss. We illustrate this case in Figure 3.7. We depict the data lay-out in a small disk array with four strings and two reliability group. The Figure 3.7 (a) shows the fault-free lay-out: the spare space is left blank, and Figure 3.7 (b) the lay-out after failure of the first disk in the fourth string. All of the original data are still there. In this compromised situation failure of the second and third string leads to data loss. In general, data loss is caused by the failure of a string, which contained two tracks from the same reliability group. Then one of these tracks must have been reconstructed on spare space, because the original data lay-out avoided this

situation on purpose. If the failed disk was located on the string which now is failing, all the data are safe. Otherwise, the probability for data loss at the track level is the product of $1/n$, (the probability that the spare track was located on the string,) with $(m-1)/m$, (the probability that the failed disks track carried a reliability group located on the string.) Globally, the situation is worse: data loss will occur with probability $1-(1-(m-1)nm)^{1000}$. For our examples, $m = 5$, $n = 10$ and $m = 10$, $n = 11$ the probabilities are $1 - 10^{-36}$ and $1 - 10^{-37}$ respectively.

If a string fails first, then at most one reliability group has still the original redundancy. In these circumstances, a disk failure will lead with even higher probability than before to data loss. Our calculations have shown that string and disk failure are not tolerated without data loss. Of course, two string failures are fatal, too.

If a disk fails, its contents are replicated on the spare space throughout the array. Even if a second disk in the same reliability group fails, the data are safe: Only those tracks of disk 1 are on disk 2 for which disk 2 was originally empty. A scenario, in which 2 disks fail in the same reliability group and in addition a third disk fails (in either the same or another reliability group,) implies data loss, as the additional disk contained $1/(mn-1)$ of the tracks of the first failed disk which are necessary to access the data of the disk which was the second to fail. We call a reliability group vulnerable if one disk in the group has failed without being replaced by a spare. We can capture the behavior of this scheme by appending an additional state $m+2$ describing two disk failures in the same group to the Markov model in Section 3.4.1.

We are now ready to define the states of the Markov model, given in Figure 3.8. State 0 describes the fault-free situation where all components function. State labels 1, 2, $\ldots, m$ give the number of vulnerable reliability groups, that is, groups with exactly one disk failure. State $m+1$ describes string failure and state $m+2$ is our exceptional state, characterized by two failed disk in the same reliability group.

The description of repair and essential component failure are straight-forward. In state 1 we observe an transition to state $m+2$, the expceptional state, with marginal probability $(n-1)\lambda$, which replaces part of the transition to the failure state. Any component failure in the exceptional state leads to data loss, as we saw above. The other transitions are the same as in the Markov model for the classic Level 5 RAID in Section 3.4.1: Failure

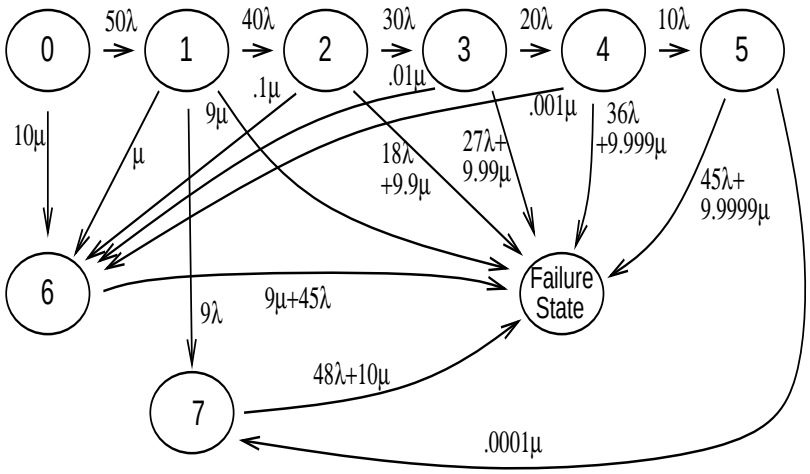


Figure 3.8: Markov Model for the Classic Raid Organization with NDS: 1 Spare, 5 Reliability Groups and 10 Strings. Not shown are transitions from repair and essential component failure. Note, that state 5 and the string failure state 6 are not identical.

of a disk in a vulnerable reliability group leads to data loss. Similarly, a string failure will lead to data loss, until all previously failed disks are located on the string.

We can strengthen reliability against disk failure further by introducing spare space to replace the contents of a second failed disk. Our scheme protects against at least three disk failures.

The Markov model is a further modification of the one in Section 3.4.1. The first two failures do not lead to more complicated read operation, because the spare space takes over. For more than three failures, we only need to keep track of how many reliability groups are vulnerable. However, if the first three failures are in exaclty two different reliability groups, data loss has been avoided: we capture this with two additional states, one of which describes two failures afflicting the same reliability group and the other one depicts three failures, two of which are in the same reliability group.

We present the Markov chains in numerical form for our small examples in Figures 3.8 and 3.10. We do not show repair and essential component failure transitions. In contrast to Figure 3.2 we did not collapse the string sailure state with state 5 (Failure of 5 disks.) In Figure 3.10 state 0 is the fault-free state, state 1 describes one lost disk, state 2 describes

two lost disk in different reliability groups, states 3, 4, 5 are labeled by the number of vulnerable reliability groups, state 6 stands for a situation in which two disks in the same reliability group have failed and state 7 for the failure of three disks, of which at least two are located in the same reliability group.

We explore the impact of reconstruction time on spare in Figures 3.9. To obtain the MTTDL values, we modify the transition rates in the Markov Chain. There is one state transition, in which we reconstruct the contents of a failed disk and store it on the spare space. We calculate the probability that an additional component failure will lead to data loss during this time and adjust the transition rate to the target state of the transition. Because the reconstruction time is almost deterministic, our procedure is better than the introduction of additional states, which would also lead to bigger matrices and thus increase the potential of numerical error. Reconstruction time impacts the MTTDL, but not greatly unless reconstruction is very leisurely. As a typical reconstruction time would be on the scale of 6 minutes to half an hour, our MTTDL times calculated on the optimistic assumptions that reconstruction is instantaneous, lead only for very large MTTDL times to noticeable differences.

Safe Distributed Sparing improves reliability considerably while incurring slightly longer disk data reconstruction times (see Chapter 10.) In SDS we do not simply write the reconstructed data on the spare space, but treat the current spare track as a member of a reliability group and perform a RAID write. While the reconstruction process is slightly more complicated, certain failure modes are excluded.

The Markov model is simple: With spare space corresponding to s disks we have the normal state $(-s)$, states $(-s+1),\ldots(0)$ describing the increasing use of spare space from one spare disk used to all spare disks used and then states (1) to (m) labelled by the number of reliability groups that are vulnerable.

The state transitions are straightforward, we only need to adjust for the fact that in states $(-s+1)$ to (m) not every reliability group contains n disks.

We give the Markov Chains for our small example RAID with one spare in Figure 3.12. We do not give the repair and essential component failure transitions. Notice, that there are two string failure states. This is necessary to distinguish a loss of a string with or without an additional failed disk.

We discuss the impact of reconstruction, repair and disk failure times in Figure 3.13. Reconstruction time becomes important for excellent MTTDL values even though the differences between SDS and NDS are not significant. Quick repair gives the RAID with SDS and superhardened strings formidable reliability. The first knee of the curve indicates the possibility of essential equipment failure. All repair curves show a steep MTTDL improvement when the repair time is lowered sufficiently. The influence of disk MTTF is not as strong. The curve for the super hardened strings flattens because string and essential component failures are becoming the main failure types.

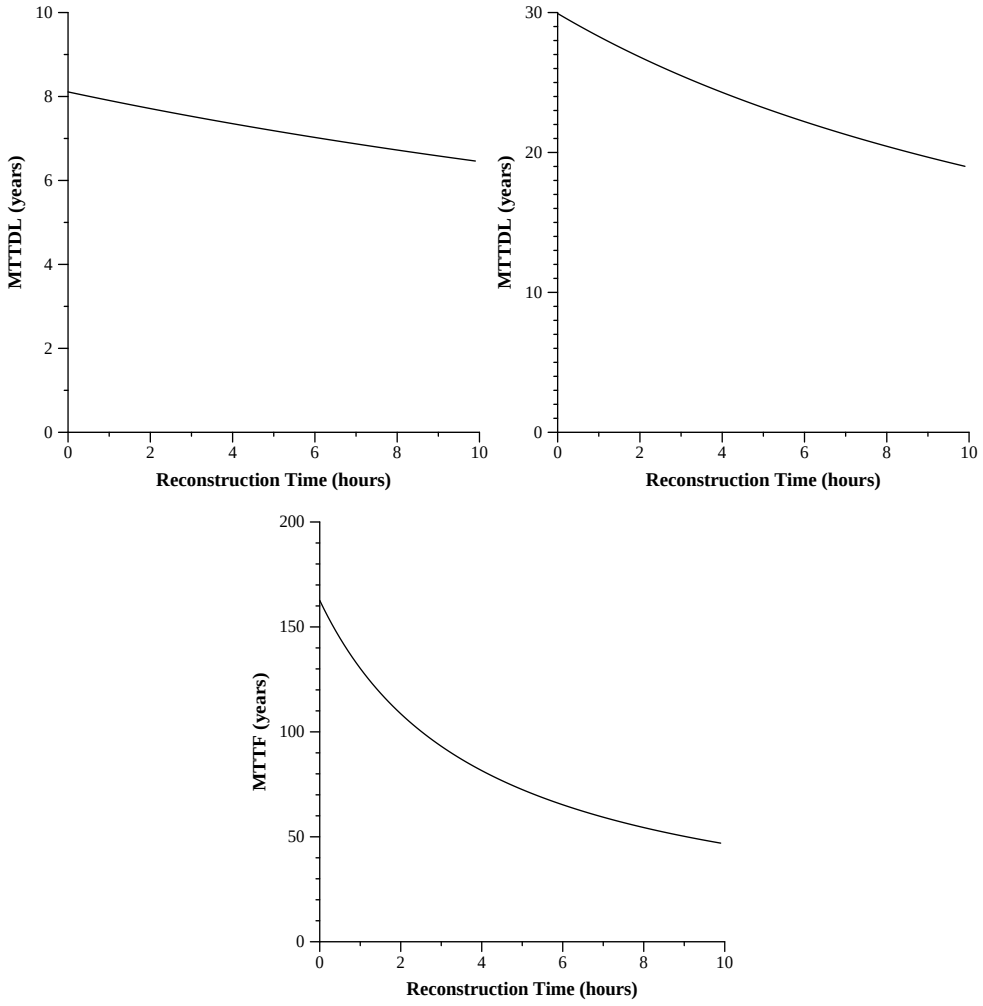


Figure 3.9: Impact of Reconstruction Time on the MTTDL for the Classic RAID with NDS (10 strings, 5 reliability groups, 1 spare)

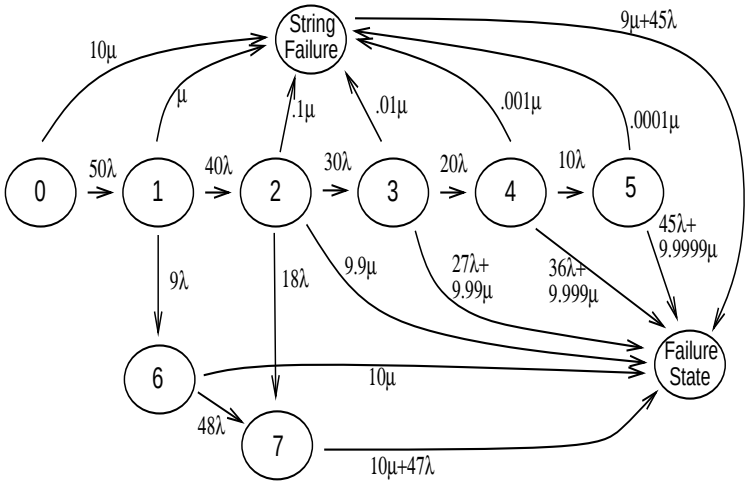


Figure 3.10: Markov Model for the Classic Raid Organization with NDS: 2 Spares, 5 Reliability Groups and 10 Strings.

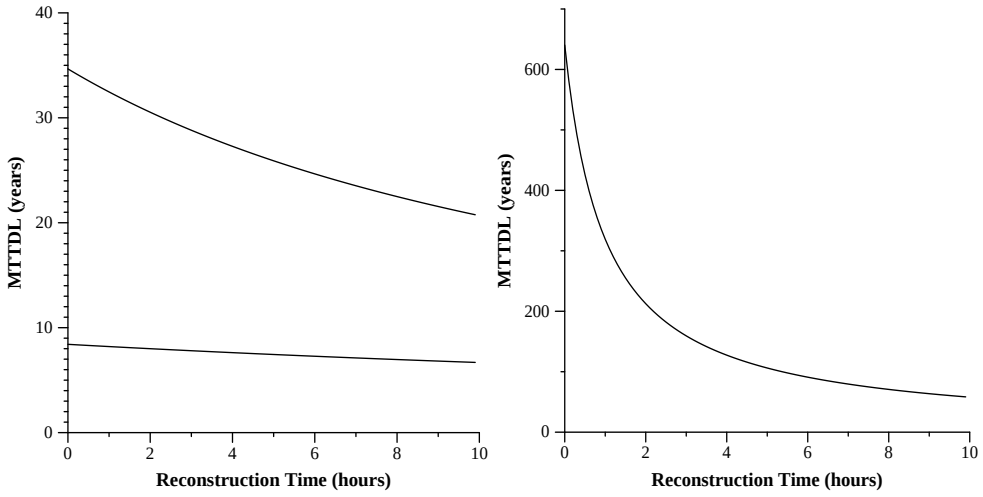


Figure 3.11: Impact of Reconstruction Time on the MTTDL for the Classic RAID with NDS (10 strings, 5 reliability groups, 2 spares)(Super Hardened Strings right.)

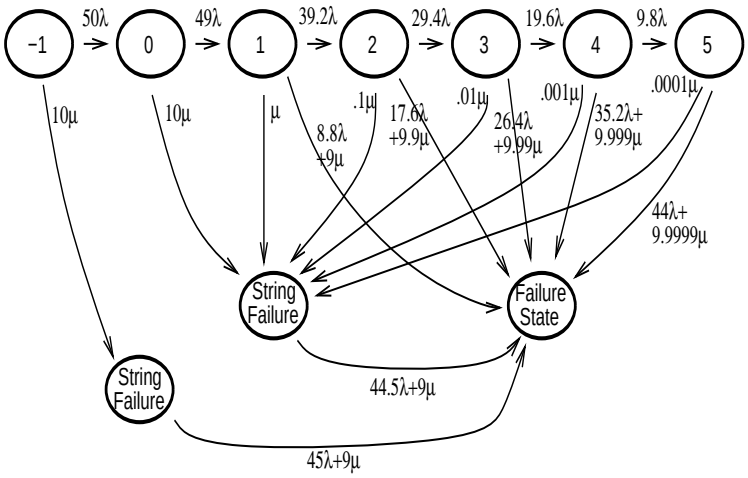


Figure 3.12: Markov Model for the Classic Raid Organization with SDS: 1 Spare, 5 Reliability Groups and 10 Strings

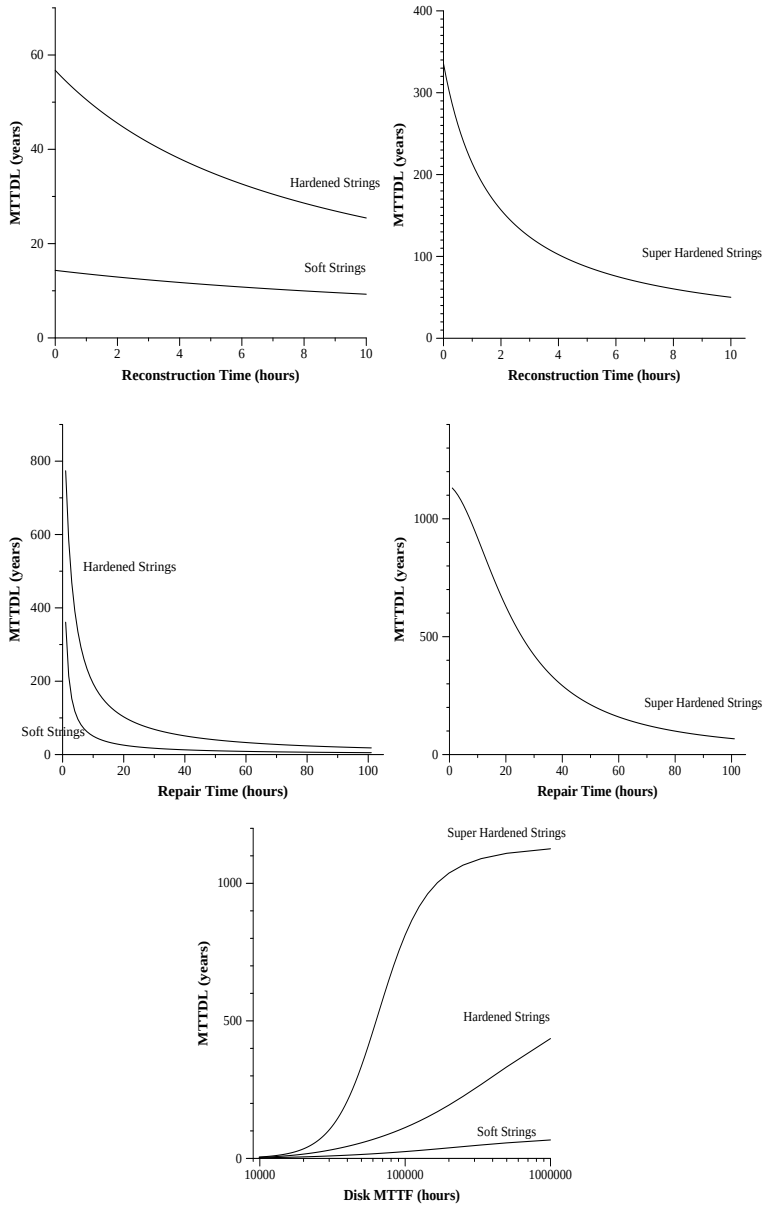


Figure 3.13: Impact of Reconstruction Time, Repair Time and Disk MTTF on the MTTDL for the Classic RAID with SDS (10 strings, 5 reliability groups, 1 spare)

(a)

track 1:		i_1	c_1
	i_2	i_2	c_2
track 2:	i_1	i_2	c_2
	i_2		c_1
track 3:	c_1	i_2	
	c_2	i_1	i_2
track 4:	i_2	c_1	i_2
	i_1	c_2	
track 5:	i_2	i_2	c_2
		i_1	c_1

(b)

track 1:	c_1	i_1	■
	i_2	i_2	c_2
track 2:	i_1	i_2	■
	i_2	c_2	c_1
track 3:	c_1	i_2	■
	c_2	i_1	i_2
track 4:	i_2	c_1	■
	i_1	c_2	i_2
track 5:	i_2	i_2	■
	c_2	i_1	c_1

Figure 3.14: A Two By Three Disk Array: (a) Fault-Free Data Lay-Out, (b) Data Lay-Out after Disk Failure

3.4.5 The Level 5 RAID with Distributed Sparing and Almost Complete Load Balancing

Distributed Sparing of even one disk increases the disk failure tolerance of an organization with load balancing to reduce the reliability difference between the classic schemes and load balancing scheme sufficiently, to make the latter ones attractive. Our results are valid for load balancing schemes, that, like ACATS, do not place two elements of the same reliability group on the same string or on the same disk during reconstruction of data from a failed unit. We first investigate the NDS scheme. The RAID tolerates string failure without data loss if no disk has failed previously. We argue string failure with previous disk loss in a small example given in Figure 3.14:

Our small example features a disk array with 2 reliability groups and 3 strings. One reliability group consist of only one information track and one check track whereas the other has two information tracks and one check track. A blank space indicates a spare space. To the right, the same disk array has suffered a disk failure, indicated by a filled-in rectangle. The spare space is used to store the reconstructed data previously stored on the now failed disk. If string three fails, on which the failed disk is located, then we do not suffer data loss. However, if string 1 fails, we can suffer data loss, as evidenced by track 5: two tracks from the same reliability group are located on string 1.

We need to investigate in more detail: We distinguish two kinds of reliability

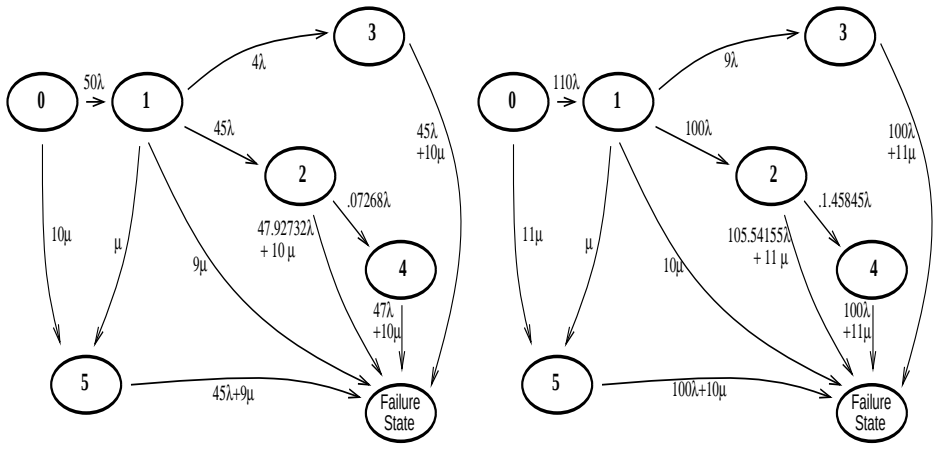


Figure 3.15: Markov Chains for Distributed Sparing with Almost Complete Load Balancing with Parameters: 1 Spare, $m = 5$, $n = 10$ and 1 Spare, $m = 10$, $n = 11$.

groups, (1) there is one reliability group which gave up one information track for use as spare space and (2), there are all the others. In Figure 3.14 the reliability group 1 exemplifies the first kind and 2 the second. If a reliability group of kind (1) has a track located on the failed disk, the security of its data did not really change; it is as if the spare track would have become inaccessible. However, a reliability group of kind (2) will become vulnerable if it looses a track due to the disk failure. This track will be reconstructed on the spare track and be located on the same string as another track of the reliability group. Failure of the string implies data loss.

The ingredients for data loss at the track level are: (a) a reliability group of kind (2) looses a track due to the disk failure and (b) the string on which the spare space was located fails. The probability that a string will lead to data loss at the track level is $\frac{m-1}{m} \cdot \frac{1}{n}$ $= (m-1)/mn$. For a thousand tracks the probability of data loss due to a string failure is $1 - (1 - (m-1)/mn)^{1000}$ which is about $1 - 10^{-38}$ for both of our examples. We will assume certainty of data loss in our subsequent calculations.

The Markov model contains six non-failure states: The fault-free state, state 0, describes no component failure, state 1 describes the situation where a disk has failed and is replaced by the spare space, state 2 portrays circumstances where two disks on different strings have failed. State 3 depicts two or more disk failures on the same string, state 4

represents one disk failure on one and two on another string and finally, state 5 describes string failure. State 4 excludes data loss, it could be omitted with only a slight loss of accuracy. States 2 and 3 could then be grouped together.

The state transitions include repair and essential component failure. From the fault-free state, a disk failure transition leads with marginal probability $nm\lambda$ to state 1, and a string failure leads with marginal probability $n\mu$ to state 5. Disk failure in state 1 happens with probability $(nm-1)\lambda$ and transitions the system to state 2 with marginal probability $(n-1)m\lambda$ and to state 3 with marginal probability $(m-1)\lambda$. As we have seen, failure of the string on which the already failed disk is located, does not lead to data loss: we have a transition from state 1 to state 5 with probability μ. The other string failures are fatal and are represented by a transition from state 1 to the failure state taken with probability $(n-1)\mu$.

We have already seen that in state 2 a string failure will lead to data loss with probability undistinguishable from 1. The prospect in case of a further disk failure are almost as grim. If the third disk failure is located on a different string than the second, we observe data loss with probability undistinguishable from 1. (The situation is even worse than the one considered in 3.4.2.) Else, the only scenario for data loss has the second or third failed disk carry the spare track originally and now a track from reliability group R and the other one a track from the same reliability group R. If the reliability group is of kind (1), this scenario cannot occur. To calculate the probability q_t for this scenario at the track level, given that we are in state 2, we argue that with probability $2/mn$ the second or third disk failure hit the spare track and that with probability $1/m$ the first failed disk carried a track in the same reliability group as the third or second failed disk. We have

$$q_t = \frac{2}{m^2 n}.$$

For our sample values $m = 5$ and $n = 10$ we have $q_t = .004$ and for $m = 10$ and $n = 11$ $q_t = .00181818$. The global picture is not as encouraging, the data loss probabilities are $q_g = 0.98183$ and $q_g = 0.83795$ respectively. To summarize, with marginal probability

$$(1 - q_g)(m-1)\lambda$$

we observe a transition from state 2 to state 4 and with marginal probability

$$\epsilon + n\mu + (q_g(m-1) + (n-1)m - 1)\,\lambda$$

a transition to the failure state.

A string failure in state 3 is fatal, if it does not afflict the already compromised string, in which case we observe a transition to state 5 with marginal probability μ. Failure of an additional disk on the same string does not change the picture. As we have seen before, failure of two disks on different strings is tantamount to data loss. We observe a transition to failure state with marginal probability $(n-1)\mu + (n-1)m\lambda$. Any further component failure in state 4 leads to data loss, we observe the corresponding transition with marginal probability $n\mu + (nm-3)\lambda$. Similarly, any further component failure in state 5 leads to data loss, the transition happens with marginal probability $(n-1)\mu + (n-1)m\lambda$.

We represent the Markov chain in numerical form for our base examples in Figure 3.15. As in the previous section, we can enhance reliability at the cost of performance by SDS: We protect reconfigured data from a failed disk by updating the check information of the reliability group in which this information was written. (Of course, if the spare track is in the same reliability group as the replaced one, then the check information is not updated.) Then the reconfigured data can only be lost if the host reliability group (which carries the spare track) suffers data loss itself. Hence, as long as there is sufficient spare space, the system tolerates disk loss and, after spare space is exhausted, behaves just like the RAID with almost complete address translation. We present the full Markov chain (including essential component failure and repair transitions) for our example RAID with one disk worth of spare space in Figure 3.17. We distinguish only four states: the fault-free state -1, state 0, which depicts loss of one disk and full use of the spare, the vulnerable state, in which one or more disks on the same string have failed, and the string failure state, which actually only describes the effects of string failure occurring in the fault-free state. The number of disks left distinguishes between the string failure and the vulnerable state. In the vulnerable state on average $(n-1)(nm-1)m/nm$ disks are left outside the compromised string, whereas the number in the string failure state is slightly higher with $(n-1)m$. We could fold both states into one with only slight inaccuracy. Again, we present a sensitivity anlysis in Figure 3.18 using the parameters in Table 2.1

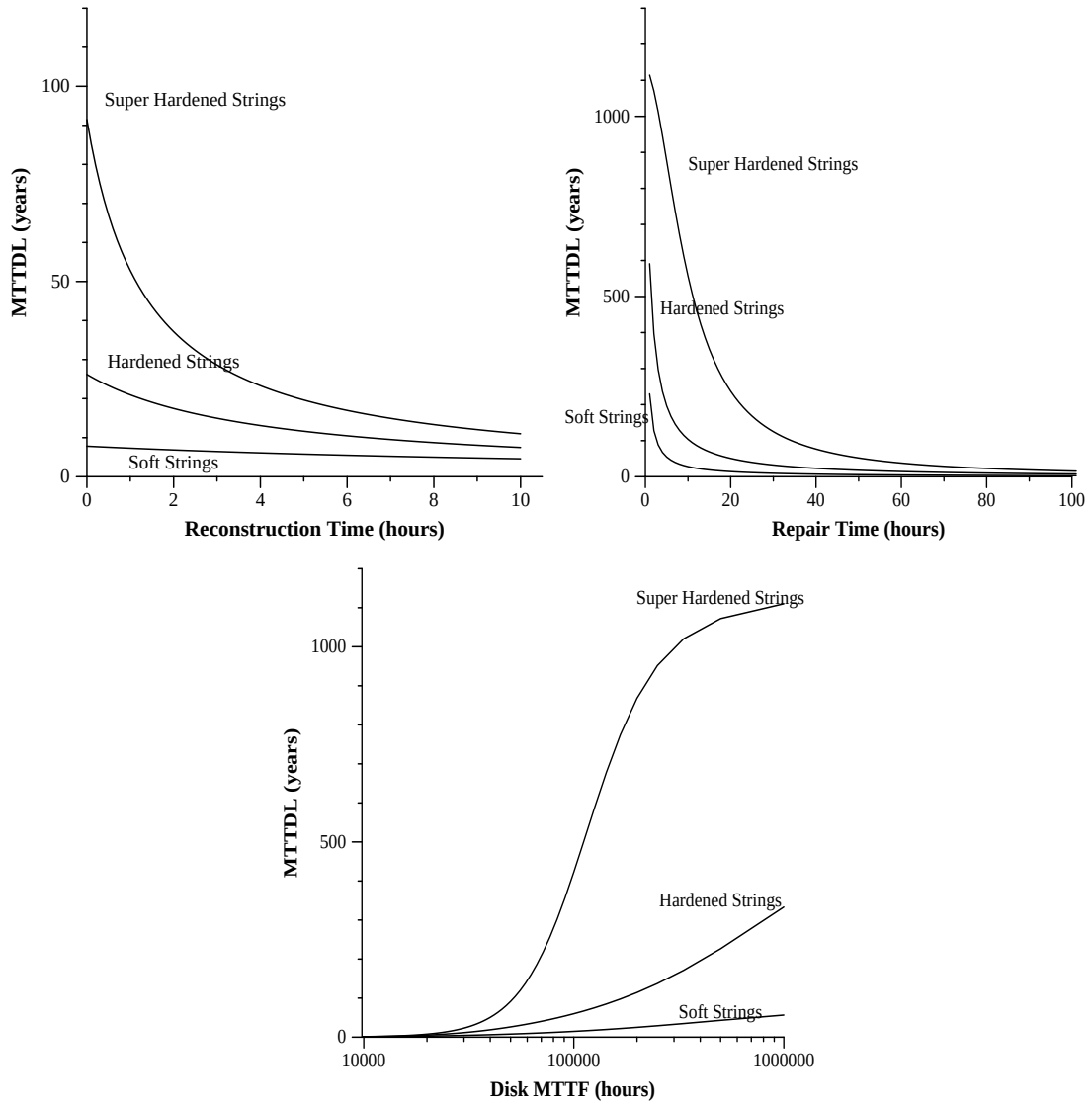


Figure 3.16: Impact of Reconstruction Time, Repair Time and Disk MTTF on the ACATS RAID with NDS (1 Spare)

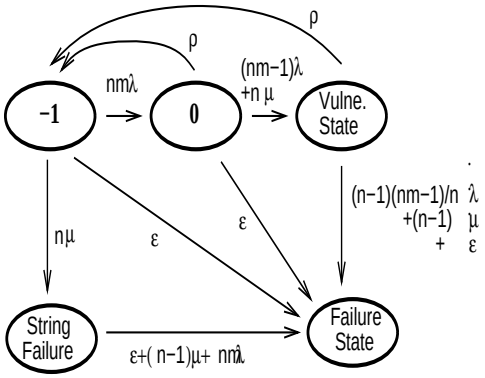


Figure 3.17: Markov Chain for Safe Distributed Sparing with Almost Complete Load Balancing with 1 Spare.

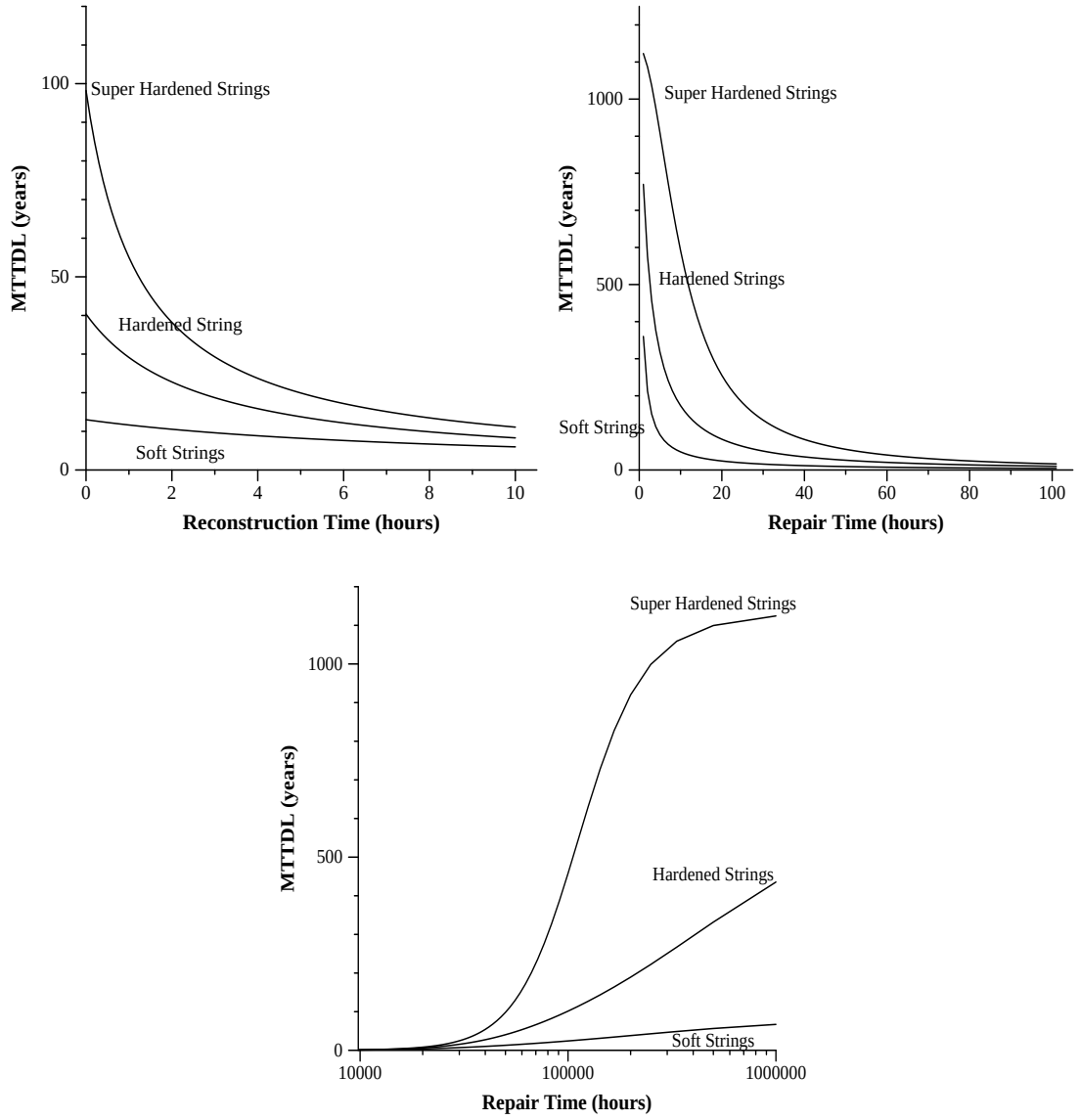


Figure 3.18: Impact of Reconstruction Time, Repair Time and Disk MTTF on the ACATS RAID with SDS (1 Spare)

3.4.6 Classic Level 5 RAID with a String of Spares

This organization shows very similar resilience as the organizations using distributed disk sparing with SDS. Our analysis uses a large Markov model; this approach reduces many difficult transition rate calculations to careful accounting, but is more susceptible to numerical error because of the matrix size in the MTTDL calculation, even though we did not observe any indications of numerical error in our calculations. In addition, the approach is not amenable to changing RAID dimensions and the resulting Markov models become successively and quickly more complicated as the one for a string of S spares is contained in the one with $S+1$ spares.

If $S = 1$ then we capture the number of vulnerable reliability groups in a two-dimensional index (i, j) where the first coordinate i indicates whether an additional disk in the reliability group that saw the first disk failure has failed ($i = 1$) or not ($i = 0$). The second coordinate j is the number of vulnerable reliability groups different from the first one. The sum $i + j$ is the total number of vulnerable reliability groups. Because the spare itself can fail, we have to add a set of states (i) reflecting the number of vulnerable reliability groups after a string failure. State 0 is to be distinguished from state (0), where the spare is unavailable, and from state (0,0), where one main disk has failed and is replaced by the spare. Three additional states depict the effects of main string failure, that is the failure of a string other than the string of spares: An unused or unusable spare defines state SF1, state SF2 describes a string failure situation where data safety depends on the spare and state SF3 indicates a string failure, where the spare provides redundance for one reliability group.

If $S = 2$ we distinguish two sub-cases (A) or (B) corresponding to the location of the first two failures in the same reliability group or not. The set of states characterizing further disk failures is given by a three dimensional index (A, i, j) or (B, i, j). The first coordinate distinguishes the location of the first two disk failures, the second coordinate i gives the number of vulnerable groups among those who use a spare disk and the third coordinate j is the number of vulnerable reliability groups among the remaining reliability groups. For example, state (A,1,2) depicts the situation where first two disks in one reliability group failed, which were replaced with the two spare disks, in addition, another disk in the same reliability group has failed and there were two other disk failures in two different reliability

groups. We now need to add states that capture the effect of either the failure of the spares' string or of a spare disk. We therefore replicate the Markov models for one spare disk and no spare string. In addition, a number of states are needed to describe the impact of string failure. We need to make distinctions as to what role the spare disks are playing: whether they are unused or unusable, necessary for the data integrity of a reliability group or merely providing redundancy.

The multitude of case distinctions renders the calculation of transitions and their marginal probabilities easy but tedious. We apply our results to the small sample RAID in Tables 3.4 and 3.5 for the case of one spare disk. As usual, we have the repair transitions taken with probability ρ from every non-failure state to the normal state. Similarly, the essential component failure is depicted in a transition with probability ϵ from each state to the failure state.

A *main string failure* failure - will either result in a transition to one of the string failure states or to the failure state. A main string failure in state 0 results in a transition to state SF3 taken with marginal probability $n\mu$. Here an arbitrary disk on the failed string is replicated on the spare. State $(0, i)$ describes vulnerability of i reliability groups different from the one which first experienced disk failure. Data loss is avoided only if in all i reliability groups the failed disk is located on the failing string. In that case, the location of the first failed disk, which was replaced by the spare disk, determines whether we need the spare disk for data safety or not. With marginal probability μ/n^i the system transitions to state SF3, with marginal probability $(n-1)\mu/n^i$ to state SF2, and with marginal probability $(1 - n^{-i+1})n\mu$ to the failure state. In state $(1, i)$ data safety depends on the location of all failed disks with respect to the failing string. If the failing string contains the first failed disk, which was replaced by the spare disk, or the other failed disk in this reliability group, then the string failure is not going to cause data loss in this reliability group and globally not with probability n^{-i}. In this case we observe a a transition to state SF2 (taken with marginal probability $2n^{-i}\mu$.) We observe alternatively a transition to the failure state with marginal probability $(n - 2n^{-1})\mu$. In state (i) the spare disk has become unusable and in addition, i groups are vulnerable. A transition to state SF1 takes place with probability $n^{-i+1}\mu$ and to the failure state with probability $(1 - n^i)n\mu$. In states SF1, SF2 and SF3 an additional main string failure will lead to data loss.

A *main disk failure* will transition from state 0 to state (0,0) with marginal probability $nm\lambda$. A main disk failure results in a transition from state $(0,i)$ to state $(1,i)$ with marginal probability $(n-1)\lambda$, to state $(0,i+1)$ with marginal probability $(m-i-1)n\lambda$ and with marginal probability $i(n-1)\lambda$ to the failure state. It causes a transition from state SF1 to the failure state with marginal probability $(n-1)m\lambda$, from state SF2 to the failure state with probability $((n-1)m-1)\lambda$, as there is one less disk to fail, and from state SF3 to the failure state with marginal probability $(n-1)(m-1)\lambda$ and from the same state to state SF2 with marginal probability $(n-1)\lambda$.

A *spare disk failure* is equivalent to *spare's string failure* and induces a state transition with probability $\lambda+\mu$. This events results in a transition from state 0 to state (0). The loss of the spare transitions the system from state $(0,i)$ into state $(i+1)$ and from state $(1,i)$ into the failure state. The event is impossible in states (i) and state SF1. In state SF3 it induces a transition to state SF2 and in states SF1 and SF2 a transition to the failure state.

We give explicit state transitions for our example in Tables 3.4 and 3.5. We use these tables for verification of our programs. The state transitions for $S=2$ are similarly given in Tables 3.6 to 3.7. The derivations again are straightforward and we do not give them here.

State Number		Description
0	0	Normal State
1	(0,0)	Main Disk Lost, Spare Used
2	(1,0)	Second Disk Failure in Same R.Group
3	(0,1)	Main Disk in Other R.Group Lost
4	(1,1)	Three Disk Failures
5	(0,2)	Three Disk Failures
6	(1,2)	Four Disk Failures
7	(0,3)	Four Disk Failures
8	(1,3)	Five Disk Failures
9	(0,4)	Five Disk Failures
10	(1,4)	Six Disk Failures
11	(0)	Spare Lost
12	(1)	Spare Lost and 1 R.Group Vulnerable
13	(2)	Spare Lost and 2 R.Groups Vulnerable
14	(3)	Spare Lost and 3 R.Groups Vulnerable
15	(4)	Spare Lost and 4 R.Groups Vulnerable
16	(5)	Spare Lost and 5 R.Groups Vulnerable
17	SF1	Main String Failure, Spare Unusable
18	SF2	Main String Failure, Spare Necessary
19	SF3	M. S. Failure, Spare Provides Redundancy

Table 3.3: States of the Markov Model for the Classic Level 5 RAID with a String of 1 Spare

From State	To State	Marginal Probability	Cause
0	1	50λ	Main Disk Failure
0	11	$\lambda + \mu$	Spare Disk or Spare String Failure
0	19	10μ	Main String Failure
1	2	9λ	Main Disk Failure
1	3	40λ	Main Disk Failure
1	12	$\lambda + \mu$	Spare Disk or Spare String Failure
1	18	9μ	Main String Failure
1	19	μ	Main String Failure
2	4	40λ	Main Disk Failure
2	18	μ	Main String Failure
2	19	μ	Main String Failure
2	FS	$9\lambda + 9\mu$	Disk or Spare String Failure
3	4	9λ	Main Disk Failure
3	5	30λ	Main Disk Failure
3	13	$\lambda + \mu$	Spare Disk or Spare String Failure
3	18	0.9μ	Main String Failure
3	19	0.1μ	Main String Failure
3	FS	$9\lambda + 9\mu$	Main Disk or Main String Failure
4	6	30λ	Main Disk Failure
4	18	0.2μ	Main String Failure
4	FS	$18\lambda + 10.8\mu$	Disk or String Failure
5	6	9λ	Main Disk Failure
5	7	20λ	Main Disk Failure
5	14	$\lambda + \mu$	Spare Disk or Spare String Failure
5	18	0.09μ	Main String Failure
5	19	0.01μ	Main String Failure
5	FS	$18\lambda + 9.9\mu$	Main Disk or Main String Failure
6	8	20λ	Main Disk Failure
6	18	$.02\mu$	Main String Failure
6	FS	$27\lambda + 10.98\mu$	Disk and String Failure
7	8	9λ	Main Disk Failure
7	9	10λ	Main Disk Failure
7	15	$\lambda + \mu$	Spare Disk or Spare String Failure
7	18	0.009μ	Main String Failure
7	19	0.001μ	Main String Failure
7	FS	$27\lambda + 9.99\mu$	Main Disk or Main String Failure

Table 3.4: State Transitions in the Markov Model for the Classic Level 5 RAID with a String of 1 Spare: Part I

From State	To State	Marginal Probability	Cause
8	10	10λ	Main Disk Failure
8	18	0.002μ	Main String Failure
8	FS	$36\lambda + 10.998\mu$	Disk or String Failure
9	10	9λ	Main Disk Failure
9	16	$\lambda + \mu$	Spare Disk or Spare String Failure
9	18	$.0009\mu$	Main String Failure
9	19	$.0001\mu$	Main String Failure
9	FS	$36\lambda + 9.999\mu$	Main Disk or Main String Failure
10	18	0.0002μ	Main String Failure
10	FS	$45\lambda + 10.9998\mu$	Disk or String Failure
11	12	50λ	Main Disk Failure
11	17	10μ	Main String Failure
12	13	40λ	Main Disk Failure
12	17	μ	Main String Failure
12	FS	$9\lambda + 9\mu$	Main Disk or Main String Failure
13	14	30λ	Main Disk Failure
13	17	0.1μ	Main String Failure
13	FS	$18\lambda + 9.9\mu$	Main Disk or Main String Failure
14	15	20λ	Main Disk Failure
14	17	0.01μ	Main String Failure
14	FS	$27\lambda + 9.99\mu$	Main Disk or Main String Failure
15	16	10λ	Main Disk Failure
15	17	0.001μ	Main String Failure
15	FS	$36\lambda + 9.999\mu$	Main Disk or Main String Failure
16	17	0.0001μ	Main String Failure
16	FS	$45\lambda + 9.9999\mu$	Main Disk or Main String Failure
17	FS	$45\lambda + 9\mu$	Main Disk or Main String Failure
18	FS	$45\lambda + 10\mu$	Disk or String Failure
19	18	$10\lambda + \mu$	Disk or Spare String Failure
19	FS	$36\lambda + 9\mu$	Disk or String Failure

Table 3.5: State Transitions in the Markov Model for the Classic Level 5 RAID with a String of 1 Spare: Part II

State Number		Description
0	0	Normal State
1	1	One Failed Disk, Data Restored on Spare
2	(A,0,0)	Two Failed Disk in Same R.Group, Spares Used
3	(A,1,0)	Second Disk Failure in Same R.Group
4	(A,0,1)	Main Disk in Other R.Group Lost
5	(A,1,1)	Four Disk Failures
6	(A,0,2)	Four Disk Failures
7	(A,1,2)	Five Disk Failures
8	(A,0,3)	Five Disk Failures
9	(A,1,3)	Six Disk Failures
10	(A,0,4)	Six Disk Failures
11	(A,1,4)	Seven Disk Failures
12	(B,0,0)	Two Failed Disks in Different R.Groups, Spares Used
13	(B,1,0)	Three Disk Failures
14	(B,0,1)	Three Disk Failures
15	(B,2,0)	Four Disk Failures
16	(B,1,1)	Four Disk Failures
17	(B,0,2)	Four Disk Failures
18	(B,2,1)	Five Disk Failures
19	(B,1,2)	Five Disk Failures
20	(B,0,3)	Five Disk Failures
21	(B,2,2)	Six Disk Failures
22	(B,1,3)	Six Disk Failures
23	(B,2,3)	Seven Disk Failures
24	(a)	One Spare Lost, All Main Disks Functional
25	(a,0,0)	1 Main Disk And One Spare Lost, Other Spare Used
26	(a,1,0)	Second Disk Failure in Same R.Group
27	(a,0,1)	Main Disk in Other R.Group Lost
28	(a,1,1)	Three Disk Failures
29	(a,0,2)	Three Disk Failures
30	(a,1,2)	Four Disk Failures
31	(a,0,3)	Four Disk Failures
32	(a,1,3)	Five Disk Failures
33	(a,0,4)	Five Disk Failures
34	(a,1,4)	Six Disk Failures

Table 3.6: States of the Markov Model for the Classic Level 5 RAID with a String of 2 Spares: Part I

State Number		Description
35	(b)	Both Spares Lost, All Main Disks Functional
36	(b,1)	Spares Lost and 1 R.Group Vulnerable
37	(b,2)	Spares Lost and 2 R.Groups Vulnerable
38	(b,3)	Spares Lost and 3 R.Groups Vulnerable
39	(b,4)	Spares Lost and 4 R.Groups Vulnerable
40	(b,5)	Spares Lost and 5 R.Groups Vulnerable
41	SF1	Main String Failure, Spares Unused or Unusable
42	SF2	M. Str. Fail., 1 Spare Unusable, 1 Necessary
43	SF3	M. Str. Fail., 1 Sp. Unusable, 1 Provides Redundancy
44	SF4	M. Str. Fail., 2 Sp. Necessary
45	SF5	M. Str. Fail., 1 Sp. Necessary, 1 Prov. Redundancy
46	SF6	M. Str. Fail., 2 Spares Provide Redundancy
47	SF7	M. Str. Fail,, 1 of 2 Spares Necessary

Table 3.7: States of the Markov Model for the Classic Level 5 RAID with a String of 2 Spares: Part II

From State	To State	Marginal Probability	Cause
0	1	50λ	Main Disk Failure
0	24	2λ	Spare Disk Failure
0	35	μ	Spare String Failure
0	46	10μ	Main String Failure
1	2	9λ	Main Disk Failure
1	12	40λ	Main Disk Failure
1	25	2λ	Spare Disk Failure
1	45	9μ	Main String Failure
1	46	μ	Main String Failure
1	36	μ	Spare String Failure
2	3	8λ	Main Disk Failure
2	4	40λ	Main Disk Failure
2	26	2λ	Spare Disk Failure
2	44	8μ	Main String Failure
2	47	2μ	Main String Failure
2	FS	μ	Spare String Failure
3	5	40λ	Main Disk Failure
3	44	3μ	Main String Failure
3	FS	$9\lambda + 8\mu$	Disk or String Failure
4	5	8λ	Main Disk Failure
4	6	30λ	Main Disk Failure
4	44	0.8μ	Main String Failure
4	47	0.2μ	Main String Failure
4	28	2λ	Spare String Failure
4	FS	$9\lambda + 10\mu$	Main Disk or String Failure
5	7	30λ	Main Disk Failure
5	44	0.3μ	Main String Failure
5	FS	$18\lambda + 10.7\mu$	Disk or String Failure
6	7	8λ	Main Disk Failure
6	8	20λ	Main Disk Failure
6	30	2λ	Spare String Failure
6	44	0.08μ	Main String Failure
6	47	0.02μ	Main String Failure
6	FS	$18\lambda + 10.9\mu$	Main Disk or String Failure
7	9	20λ	Main Disk Failure
7	44	0.03μ	Main String Failure
7	FS	$27\lambda + 10.97\mu$	Disk or String Failure

Table 3.8: State Transitions in the Markov Model for the Classic Level 5 RAID with a String of 2 Spares: Part I

From State	To State	Marginal Probability	Cause
8	9	8λ	Main Disk Failure
8	10	10λ	Main Disk Failure
8	32	2λ	Spare String Failure
8	44	0.008μ	Main String Failure
8	47	0.002μ	Main String Failure
8	FS	$27\lambda + 10.99\mu$	Main Disk or String Failure
9	11	10λ	Main Disk Failure
9	44	0.003μ	Main String Failure
9	FS	$36\lambda + 10.997\mu$	Disk or String Failure
10	11	8λ	Main Disk Failure
10	34	2λ	Spare String Failure
10	44	0.0008μ	Main String Failure
10	47	0.0002μ	Main String Failure
10	FS	$36\lambda + 10.999\mu$	Disk or String Failure
11	44	0.0003μ	Main String Failure
11	FS	$45\lambda + 10.9997\mu$	Disk or String Failure
12	13	18λ	Main Disk Failure
12	14	30λ	Main Disk Failure
12	44	8.1μ	Main String Failure
12	45	1.8μ	Main String Failure
12	46	0.1μ	Main String Failure
12	27	2λ	Spare Disk Failure
12	37	μ	Spare String Failure
13	15	9λ	Main Disk Failure
13	16	30λ	Main Disk Failure
13	44	1.8μ	Main String Failure
13	45	0.2μ	Main String Failure
13	FS	$9\lambda + 9\mu$	Disk or String Failure
14	16	18λ	Main Disk Failure
14	17	20λ	Main Disk Failure
14	29	2λ	Spare Disk Failure
14	38	μ	Spare String Failure
14	44	0.81μ	Main String Failure
14	45	0.18μ	Main String Failure
14	46	0.01μ	Main String Failure
14	FS	$9\lambda + 9\mu$	Main Disk or Main String Failure

Table 3.9: State Transitions in the Markov Model for the Classic Level 5 RAID with a String of 2 Spares: Part II

From State	To State	Marginal Probability	Cause
15	18	30λ	Main Disk Failure
15	44	0.4μ	Main String Failure
15	FS	$18\lambda + 10.6\mu$	Disk or String Failure
16	18	9λ	Main Disk Failure
16	19	20λ	Main Disk Failure
16	30	λ	Spare Disk Failure
16	44	0.18μ	Main String Failure
16	45	0.02μ	Main String Failure
16	FS	$18\lambda + 10.8\mu$	Disk or String Failure
17	19	18λ	Main Disk Failure
17	20	10λ	Main Disk Failure
17	32	2λ	Spare Disk Failure
17	39	μ	Spare String Failure
17	44	0.081μ	Main String Failure
17	45	0.018μ	Main String Failure
17	46	0.001μ	Main String Failure
17	FS	$18\lambda + 9.9\mu$	Main Disk or Main String Failure
18	21	20λ	Main Disk Failure
18	44	0.04μ	Main String Failure
18	FS	$27\lambda + 10.96\mu$	Disk or String Failure
19	21	9λ	Main Disk Failure
19	22	10λ	Main Disk Failure
19	32	λ	Spare String Failure
19	44	0.018μ	Main String Failure
19	45	0.002μ	Main String Failure
19	FS	$27\lambda + 10.98\mu$	Disk or String Failure
20	22	18λ	Main Disk Failure
20	33	2λ	Spare Disk Failure
20	40	μ	Spare String Failure
20	44	0.0081μ	Main String Failure
20	45	0.0018μ	Main String Failure
20	46	0.0001μ	Main String Failure
20	FS	$27\lambda + 9.99\mu$	Main Disk or Main String Failure
21	23	10λ	Main Disk Failure
21	44	0.004μ	Main String Failure
21	FS	$36\lambda + 10.996\mu$	Disk or String Failure

Table 3.10: State Transitions in the Markov Model for the Classic Level 5 RAID with a String of 2 Spares: Part III

From State	To State	Marginal Probability	Cause
22	23	9λ	Main Disk Failure
22	34	λ	Spare String Failure
22	44	0.0018μ	Main String Failure
22	45	0.0002μ	Main String Failure
22	FS	$36\lambda + 10.998\mu$	Disk or String Failure
23	44	0.0004μ	Main String Failure
23	FS	$45\lambda + 10.9996\mu$	Disk or String Failure
24	25	50λ	Main Disk Failure
24	35	$\lambda + \mu$	Spare Disk or Spare String Failure
24	43	10μ	Main String Failure
25	26	9λ	Main Disk Failure
25	27	40λ	Main Disk Failure
25	36	$\lambda + \mu$	Spare Disk or Spare String Failure
25	42	9μ	Main String Failure
25	43	μ	Main String Failure
26	28	40λ	Main Disk Failure
26	42	μ	Main String Failure
26	43	μ	Main String Failure
26	FS	$9\lambda + 9\mu$	Disk or Spare String Failure
27	28	9λ	Main Disk Failure
27	29	30λ	Main Disk Failure
27	37	$\lambda + \mu$	Spare Disk or Spare String Failure
27	42	0.9μ	Main String Failure
27	43	0.1μ	Main String Failure
27	FS	$9\lambda + 9\mu$	Main Disk or Main String Failure
28	30	30λ	Main Disk Failure
28	42	0.2μ	Main String Failure
28	FS	$18\lambda + 10.8\mu$	Disk or String Failure
29	30	9λ	Main Disk Failure
29	31	20λ	Main Disk Failure
29	38	$\lambda + \mu$	Spare Disk or Spare String Failure
29	42	0.09μ	Main String Failure
29	43	0.01μ	Main String Failure
29	FS	$18\lambda + 9.9 * mu$	Main Disk or Main String Failure
30	32	20λ	Main Disk Failure
30	42	$.02\mu$	Main String Failure
30	FS	$27\lambda + 10.98\mu$	Disk and String Failure

Table 3.11: State Transitions in the Markov Model for the Classic Level 5 RAID with a String of 2 Spares: Part IV

From State	To State	Marginal Probability	Cause
31	32	9λ	Main Disk Failure
31	33	10λ	Main Disk Failure
31	39	$\lambda + \mu$	Spare Disk or Spare String Failure
31	42	0.009μ	Main String Failure
31	43	0.001μ	Main String Failure
31	FS	$27\lambda + 9.99\mu$	Main Disk or Main String Failure
32	34	10λ	Main Disk Failure
32	42	0.002μ	Main String Failure
32	FS	$36\lambda + 10.998\mu$	Disk or String Failure
33	34	9λ	Main Disk Failure
33	40	$\lambda + \mu$	Spare Disk or Spare String Failure
33	42	$.0009\mu$	Main String Failure
33	43	$.0001\mu$	Main String Failure
33	FS	$36\lambda + 9.999\mu$	Main Disk or Main String Failure
34	42	0.0002μ	Main String Failure
34	FS	$45\lambda + 10.9998\mu$	Disk or String Failure
35	36	50λ	Main Disk Failure
35	41	10μ	Main String Failure
36	37	40λ	Main Disk Failure
36	41	μ	Main String Failure
36	FS	$9\lambda + 9\mu$	Main Disk or Main String Failure
37	38	30λ	Main Disk Failure
37	41	0.1μ	Main String Failure
37	FS	$18\lambda + 9.9\mu$	Main Disk or Main String Failure
38	39	20λ	Main Disk Failure
38	41	0.01μ	Main String Failure
38	FS	$27\lambda + 9.99\mu$	Main Disk or Main String Failure
39	40	10λ	Main Disk Failure
39	41	0.001μ	Main String Failure
39	FS	$36\lambda + 9.999\mu$	Main Disk or Main String Failure
40	41	0.0001μ	Main String Failure
40	FS	$45\lambda + 9.9999\mu$	Main Disk or Main String Failure
41	FS	$45\lambda + 9\mu$	Main Disk or Main String Failure
42	FS	$45\lambda + 10\mu$	Disk or String Failure
43	41	$\lambda + \mu$	Spare Disk or String Failure
43	42	9λ	Main Disk Failure
43	FS	$36\lambda + 9\mu$	Disk or String Failure
44	FS	$45\lambda + 10\mu$	Disk or String Failure

Table 3.12: State Transitions in the Markov Model for the Classic Level 5 RAID with a String of 2 Spare: Part V

From State	To State	Marginal Probability	Cause
45	44	10λ	String Failure
45	FS	$36\lambda + 10\mu$	Disk or String Failure
46	41	$\lambda + \mu$	Spare Disk or Spare String Failure
46	44	9λ	Main Disk Failure
46	FS	$36\lambda + 10\mu$	Disk or String Failure

Table 3.13: State Transitions in the Markov Model for the Classic Level 5 RAID with a String of 2 Spare: Part VI

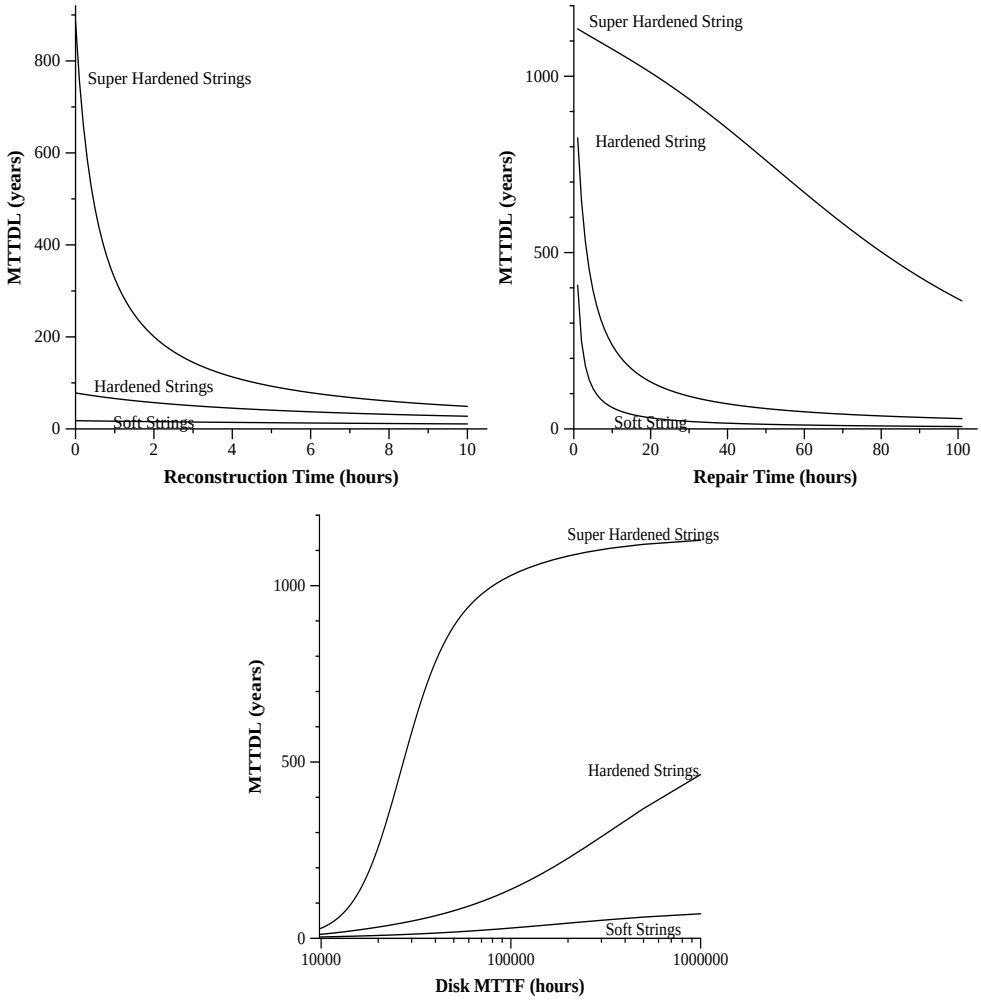


Figure 3.19: Impact of Reconstruction Time, Repair Time, Disk MTTF on the MTTDL of the RAID with ACATS and Distributed String Sparing.

3.4.7 RAID with a String of Spares and Almost Complete Load Balancing

Almost Complete Load Balancing in the form of ACATS gives us performance advantages in the disk failure case, while the presence of spare disks presents us with good failure reliability.

Preliminary Calculations and Implementation of Assignment of Spares:
We assume that the number of spares is smaller than the number of disks in a main string. Even if in the event of a main string failure all spares are used to regenerate data on this string, there is no noticeable improvement of data security over a lazy procedure, which does not use reconfiguration. We validate this with our two examples assuming that we have two spares. The disk array with 10 strings with 5 disks sees its data integrity maintained in the event of a string failure followed by a disk failure with probability 0 without use of spare disks and with probability 10^{-320} with use of spare disks to replace some of the failed strings information. The numbers for the larger array are 0.0 and 10^{-700} instead. These numbers are based on a random addressing scheme, whereas - of course - we only use a fixed addressing scheme, whose disk assignments look random. We have to interpret our numbers as saying that it would be very difficult to design an addressing scheme, for which the use of spares to replace some of the disks of a failed string noticeably improves reliability, while not at the same time limit the performance gains of the scheme.

The impact of two disk failures in the same string is far less severe than the impact of two disk failures in different strings. If we assume, that no spares are available to replace data, then we can say, that in the first case data loss cannot eoccur, whereas in the latter case data loss is practically unavoidable. To substantiate this statement, we use our two examples. In the 5×10 disk array, the data survival probability is approx. 10^{-97} and in the 10×11 disk array, it is 10^{-46}. But as we are using a concrete disk addressing scheme, these number really denote certainty.

Because of this fact, our scheme does not use the naive spare assignment policy, which assigns spares to replace failed disk in order and then never changes this assignment. True enough, reassigning spares impacts performance negatively, but as the reassignment period can be relatively long, this impact is limited. Our reassignment policy is to minimize the number of strings, which are impacted by data failure, without a spare replacing the lost disk's data.

The implementation of this scheme is not difficult because component failures occur one at a time. As we have seen, a main string failure cannot be addressed successfully with spares. We assign spares to replace string information only to improve performance and limit the extent of data loss, not its chance of occurring, in the event of further failures. For disk failures, we first use spares to replace lost data until all spare space is used. Occasionally, survival chances are increased by reassigning spares.

The Markov Model

We capture the impact of disk failures by using a "Failure Pattern" to encode the relevant information. For example, the symbol $\boxed{\frac{110}{003}}$ describes a situation with five disk failures. The upper line gives the location of failed disks which are replaced by spares, the lower line the one of failed disks not replaced by spares. Columns indicate strings, but as the order of the string is not relevant, we order strings first by the number of spares used and then by the number of failed disks without replaced data. In our example, there are 2 strings with one disk failure and one with 3 disk failures. The latter ones' data are not reconfigured on spares. As these are in one string, data loss has not occurred. The lower string representing disk failures without replacement by spares can only contain one nonzero number. To illustrate our procedure, an explicit example is calculated in Table 3.14 and 3.15 for the 5×10 RAID with a string of two spares. Failure Patterns are only sufficient to describe main disk failure, to indicate spare disk failure, we amend the descriptor by the number of failed spare disks. Thus $\boxed{\frac{0}{0} \mid 1}$ describes the failure of one spare disk.

For certain failure patterns we will reassign the spares to maximize resilience against further disk loss. We capture these cases in Table 3.16 for our example. The improvement is marginal, even for larger number of spares, as the system enters the relevant states only seldom.

We present the results of a sensitivity analysis in Figures 3.20 and 3.21. The parameters are given in Table 2.1.

State Number		Description
0	$\frac{0}{0}$	Normal State
1	$\frac{1}{0}$	1 Failed Main Disk
2	$\frac{2}{0}$	2 Failed Main Disks In Same String
3	$\frac{11}{00}$	2 Failed Main Disks In Different Strings
4	$\frac{2}{1}$	3 Failed Main Disks
5	$\frac{20}{01}$	3 Failed Main Disks
6	$\frac{110}{001}$	3 Failed Main Disks
7	$\frac{2}{2}$	4 Failed Main Disks
8	$\frac{20}{02}$	4 Failed Main Disks
9	$\frac{11}{20}$	4 Failed Main Disks
10	$\frac{110}{002}$	4 Failed Main Disks
11	$\frac{2}{3}$	5 Failed Main Disks
12	$\frac{20}{03}$	5 Failed Main Disks
13	$\frac{11}{30}$	5 Failed Main Disks
14	$\frac{110}{003}$	5 Failed Main Disks
15	$\frac{20}{04}$	6 Failed Main Disks
16	$\frac{11}{40}$	6 Failed Main Disks
17	$\frac{110}{004}$	6 Failed Main Disks
18	$\frac{20}{05}$	7 Failed Main Disks
19	$\frac{110}{005}$	7 Failed Main Disks

Table 3.14: States of the Markov Model for the Level 5 RAID with a String of 2 Spares and Almost Complete Load Balancing

State Number		Description
20	$\left[\frac{0}{0} \mid 1\right]$	1 Spare Disk Failure
21	$\left[\frac{1}{0} \mid 1\right]$	1 Spare and 1 Main Disk Lost
22	$\left[\frac{1}{1} \mid 1\right]$	1 Spare and 2 Main Disks Lost
23	$\left[\frac{10}{01} \mid 1\right]$	1 Spare and 2 Main Disks Lost
24	$\left[\frac{1}{2} \mid 1\right]$	1 Spare and 3 Main Disks Lost
25	$\left[\frac{10}{02} \mid 1\right]$	1 Spare and 3 Main Disks Lost
26	$\left[\frac{1}{3} \mid 1\right]$	1 Spare and 4 Main Disks Lost
27	$\left[\frac{10}{03} \mid 1\right]$	1 Spare and 4 Main Disks Lost
28	$\left[\frac{1}{4} \mid 1\right]$	1 Spare and 5 Main Disks Lost
29	$\left[\frac{10}{04} \mid 1\right]$	1 Spare and 5 Main Disks Lost
30	$\left[\frac{10}{05} \mid 1\right]$	1 Spare and 6 Main Disks Lost
31	$\left[\frac{0}{0} \mid 2\right]$	2 Spares Lost
32	$\left[\frac{0}{1} \mid 2\right]$	2 Spares And 1 Main Disk Lost
33	$\left[\frac{0}{2} \mid 2\right]$	2 Spares And 2 Main Disk Lost
34	$\left[\frac{0}{3} \mid 2\right]$	2 Spares And 3 Main Disk Lost
35	$\left[\frac{0}{4} \mid 2\right]$	2 Spares And 4 Main Disk Lost
36	$\left[\frac{0}{5} \mid 2\right]$	2 Spares And 5 Main Disk Lost
37	FMS	Failed Main String, 2 Spares necessary
38	FMS	Failed Main String, 1 Spare necessary
39	FMS	Failed Main String, No Spares used

Table 3.15: States of the Markov Model for the Level 5 RAID with a String of 2 Spares and Almost Complete Load Balancing

Original Failure Pattern	New State Number	New Failure Patterns
$\left[\frac{11}{10}\right]$	5	$\left[\frac{20}{01}\right]$

Table 3.16: Failure Patterns of the Markov Model for the Level 5 RAid with a String of 2 Spares and Almost Complete Load Balancing, that are not realized due to Reassignment of Spares.

From State	To State	Marginal Probability	Cause
0	1	50λ	Main Disk Failure
0	20	2λ	Spare Disk Failure
0	39	11μ	String Failure
1	2	4λ	Main Disk Failure
1	3	45λ	Main Disk Failure
1	21	2λ	Spare Disk Failure
1	32	μ	Spare String Failure
1	38	9μ	Main String Failure
1	39	μ	Main String Failure
2	4	3λ	Main Disk Failure
2	5	45λ	Main Disk Failure
2	22	2λ	Spare Disk Failure
2	33	μ	Spare String Failure
2	37	9μ	Main String Failure
2	39	μ	Main String Failure
3	5	8λ	Main Disk Failure
3	6	40λ	Main Disk Failure
3	23	2λ	Spare Disk Failure
3	37	8μ	Main String Failure
3	38	2μ	Main String Failure
3	FS	μ	Spare String Failure
4	7	2λ	Main Disk Failure
4	24	2λ	Spare Disk Failure
4	34	μ	Spare String Failure
4	37	9μ	Main String Failure
4	39	μ	Main String Failure
5	8	4λ	Main Disk Failure
5	FS	$45\lambda + 11\mu$	Disk And String Failure
6	10	4λ	Main Disk Failure
6	37	μ	Main String Failure
6	FS	$45\lambda + 10\mu$	Disk or String Failure
7	11	λ	Main Disk Failure
7	26	2λ	Spare Disk Failure
7	35	μ	Spare String Failure
7	39	μ	Main String Failure
7	FS	$45\lambda + 9\mu$	Main Disk or Main String Failure

Table 3.17: State Transitions in the Markov Model for the Classic Level 5 RAID with a String of 1 Spare: Part I

From State	To State	Marginal Probability	Cause
8	12	3λ	Main Disk Failure
8	37	μ	Main String Failure
8	39	μ	Spare String Failure
8	FS	$45\lambda + 9\mu$	Disk or Main String Failure
9	13	2λ	Main Disk Failure
9	27	λ	Spare Disk Failure
9	38	μ	Main String Failure
9	FS	$45\lambda + 10\mu$	Disk or String Failure
10	14	3λ	Main Disk Failure
10	37	μ	Main String Failure
10	FS	$45\lambda + 10\mu$	Disk or String Failure
11	28	2λ	Spare Disk Failure
11	39	2μ	Main or Spare String Failure
11	FS	$45\lambda + 9\mu$	Main Disk or Main String Failure
12	15	2λ	Main Disk Failure
12	37	μ	Main String Failure
12	FS	$45\lambda + 10\mu$	Disk or String Failure
13	16	λ	Main Disk Failure
13	29	λ	Spare Disk Failure
13	38	μ	Main String Failure
13	FS	$45\lambda + 10\mu$	Disk or String Failure
14	17	2λ	Main Disk Failure
14	37	μ	Main String Failure
14	FS	$45\lambda + 10\mu$	Disk or String Failure
15	18	λ	Main Disk Failure
15	37	μ	Main String Failure
15	FS	$45\lambda + 11\mu$	Disk or String Failure
16	30	λ	Spare Disk Failure
16	38	μ	Main String Failure
16	FS	$45\lambda + 10\mu$	Disk or String Failure
17	19	λ	Main Disk Failure
17	37	μ	Main String Failure
17	FS	$45\lambda + 10\mu$	Disk or String Failure
18	37	μ	Main String Failure
18	FS	$45\lambda + 10\mu$	Disk or String Failure
19	37	μ	Main String Failure
19	FS	$45\lambda + 10\mu$	Disk or String Failure

Table 3.18: State Transitions in the Markov Model for the Classic Level 5 RAID with a String of 1 Spare: Part II

From State	To State	Marginal Probability	Cause
20	21	50λ	Main Disk Failure
20	31	$\lambda + \mu$	Spare Disk or Spare String Failure
20	39	10μ	Main String Failure
21	22	4λ	Main Disk Failure
21	23	45λ	Main Disk Failure
21	32	$\lambda + \mu$	Spare Disk or Spare String Failure
21	37	μ	Main String Failure
21	38	9μ	Main String Failure
22	24	3λ	Main Disk Failure
22	33	$\lambda + \mu$	Spare Disk or Spare String Failure
22	37	μ	Main String Failure
22	FS	$45\lambda + 9\mu$	Main Disk or Main String Failure
23	25	4λ	Main Disk Failure
23	38	μ	Main String Failure
23	FS	$45\lambda + 10\mu$	Disk or String Failure
24	26	2λ	Main Disk Failure
24	34	$\lambda + \mu$	Spare Disk or Spare String Failure
24	37	μ	Main String Failure
24	FS	$45\lambda + 9\mu$	Main Disk or Main String Failure
25	27	3λ	Main Disk Failure
25	38	μ	Main String Failure
25	FS	$45\lambda + 10\mu$	Disk or String Failure
26	28	λ	Main Disk Failure
26	35	$\lambda + \mu$	Spare Disk or Spare String Failure
26	37	μ	Main String Failure
26	FS	$45\lambda + 9\mu$	Main Disk or Main String Failure
27	29	2λ	Main Disk Failure
27	38	μ	Main String Failure
27	FS	$45\lambda + 10\mu$	Disk or String Failure
28	36	$\lambda + \mu$	Spare Disk or Spare String Failure
28	37	μ	Main String Failure
28	FS	$45\lambda + 9\mu$	Main Disk or Main String Failure
29	30	λ	Main Disk Failure
29	38	μ	Main String Failure
29	FS	$45\lambda + 10\mu$	Disk or String Failure

Table 3.19: State Transitions in the Markov Model for the Classic Level 5 RAID with a String of 1 Spare: Part III

From State	To State	Marginal Probability	Cause
30	38	μ	Main String Failure
30	FS	$45\lambda + 10\mu$	Disk or String Failure
31	32	50λ	Main Disk Failure
31	39	10μ	Main String Failure
32	33	4λ	Main Disk Failure
32	39	μ	Main String Failure
32	FS	$45\lambda + 9\mu$	Main Disk or Main String Failure
33	34	3λ	Main Disk Failure
33	39	μ	Main String Failure
33	FS	$45\lambda + 9\mu$	Main Disk or Main String Failure
34	35	2λ	Main Disk Failure
34	39	μ	Main String Failure
34	FS	$45\lambda + 9\mu$	Main Disk or Main String Failure
35	36	λ	Main Disk Failure
35	39	μ	Main String Failure
35	FS	$45\lambda + 9\mu$	Main Disk or Main String Failure
36	FS	$45\lambda + 9\mu$	Main Disk Failure
37	FS	$45\lambda + 10\mu$	Main Disk or String Failure
38	FS	$45\lambda + 10\mu$	Disk or String Failure
39	FS	$45\lambda + 9\mu$	Disk or Main String Failure

Table 3.20: State Transitions in the Markov Model for the Classic Level 5 RAID with a String of 1 Spare: Part IV

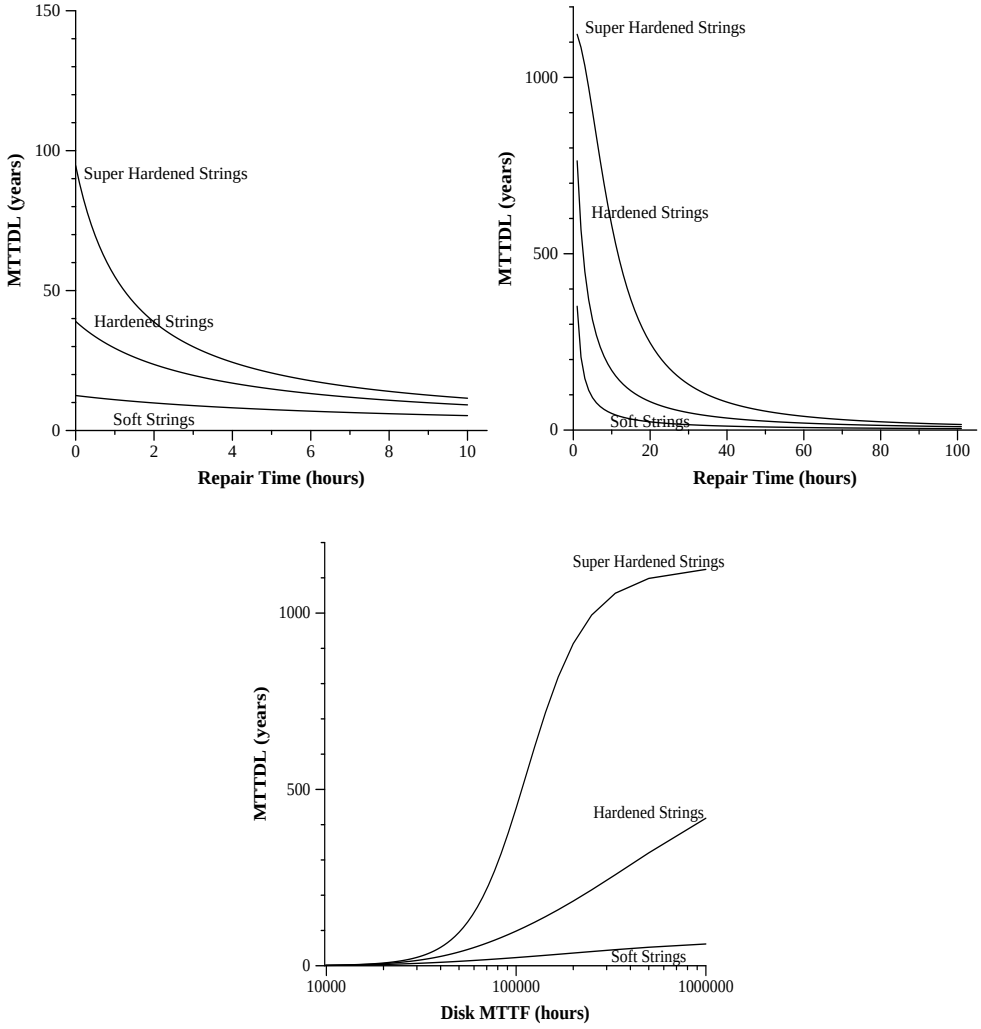


Figure 3.20: Impact of Reconstruction Time, Repair Time, Disk MTTF on the MTTDL of the RAID with ACATS and a String of one Spare.

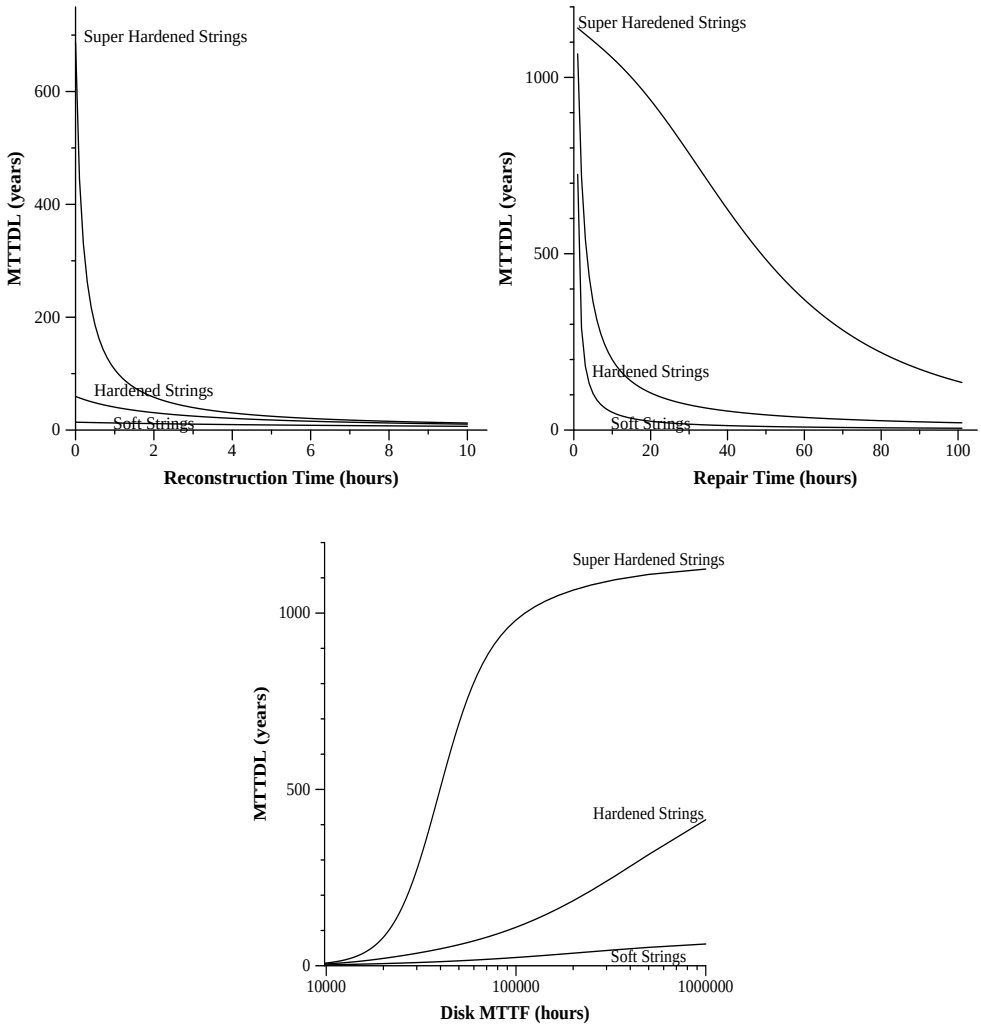


Figure 3.21: Impact of Reconstruction Time, Repair Time, Disk MTTF on the MTTDL of the RAID with ACATS and a String of two Spares.

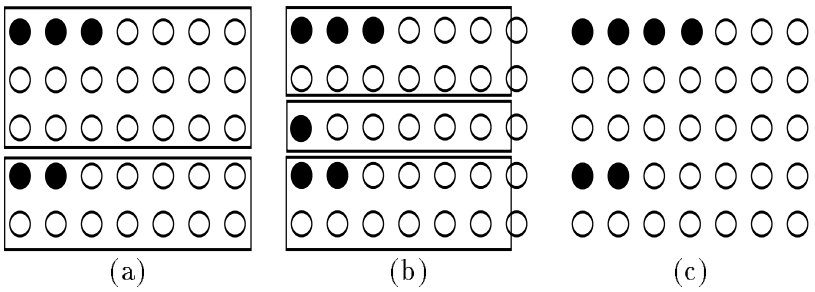


Figure 3.22: Partitioning Example

3.4.8 Classical Level 5 RAID with a Distributed Spare String

(a) Spare Assignment Policy: Our organization consists of n strings with m disks. At the logical level, one of the strings is a check string, another is the string of spares and the remaining $n-2$ are information strings. We use our standard addressing scheme and permute disk addresses inside a reliability group. We can use a single cyclic permutation. The redundancy is high enough to guarantee data integrity against two string failures, against one string and one disk failure and against at least $m+1$ disk failures.

We achieve this redundancy using a *generous* spare allocation policy. If a disk fails, we use the spare space in the reliability group in which the disk is located to reconstruct its data. If a second disk in the same reliability group fails, we use the spare space of another reliability group to hold its data and so on. If, however, a disk in a reliability group, that "loaned" its spare space to another group, is lost, then we "call in the loan" and reconfigure the data of this last disk on the local spare space and the previously hold data on another free spare space. This seemingly complicated clause increases the resilience against disk failure and really involves at most copying the spare space contents before we overwrite them with new reconstructed data.

(b) Partitions: The first m failed disks see their data replaced in the spare space. If x_1 disks among these are located in the first reliability group, then the spare space of x_i reliability groups is used for the replacement of the disks' data. We accordingly partition the reliability groups in *partitions;* the spare space of the members of each partition is used to replace disk data of one of the members of the partition, which we call the leader of the

(a)

track 1:	a_1	a_2	a_3	
	b_1	b_2	b_3	
	c_1	c_2	c_3	
track 2:		a_1	a_2	a_3
		b_1	b_2	b_3
		c_1	c_2	c_3
track 3:	a_3		a_1	a_2
	b_3		b_1	b_2
	c_3		c_1	c_2

(b)

track 1:	a_1	■	■	■
	b_1	b_2	b_3	a_3
	c_1	c_2	c_3	a_2
track 2:	a_3	■	■	■
	a_2	b_1	b_2	b_3
	a_1	c_1	c_2	c_3
track 3:	a_3	■	■	■
	b_3	a_2	b_1	b_2
	c_3	a_3	c_1	c_2

Figure 3.23: A Three by Four Disk Array: (a) Normal Data Lay-Out (b) Data Lay-Out after 3 Disk Failures

partition. (Loosely, we talk of a partitioning of the disk array.) We can always simplify our illustrations by assuming that the partitions consist of neighboring reliability groups with the leader being the highest, though we use of course a first-come-first-served policy. With additional disk failures, partitions become "vulnerable" and are subject to further subdivision.

We illustrate partitioning in Figure 3.4.8. The left figure depicts 5 disk failure in a disk array with 5 reliability groups. Three of these disk failures were located in one and two in another reliability group. These groups, together with the ones whose spare space used to replace the failed disks, make up the two partitions of the array, represented symbolically by the two boxes. The next two figures show the effect of an additional disk failure in the larger partition. If the disk failure is in another reliability group (Middle of Figure 3.4.8) the partition is subdivided. Otherwise, however, the big partition is retained. (Right of Figure 3.4.8).

We illustrate our discussion in Figure 3.23. Here we show the data lay-out in a small disk array with three reliability groups of four disks each. Part (a) illustrates the normal lay-out and part (b) after the failure of three disks.

In general, if x_i disks have failed in reliability group i, it becomes the leader of a partition of x_i reliability groups. If a further disk in the same reliability group fails, and if no additional spare space can be allocated, then the reliability group becomes *vulnerable,* one step removed from data loss. In our example, all disks in the first reliability group will

be lost. We observe, that any further disk loss in the partition will actually lead to data loss in the first reliability group.

If instead the first additional disk failure is in another reliability group in the same partition, then the partition is split into two; one of the two partitions will consist of just the reliability group with the additional disk failure and the other one encompasses all the other groups. The original leader is now vulnerable, as it lost one disk worth of spare space. We can observe, that any further disk loss in the partition will loose some of the leader's reconstructed data and lead to data loss. (See Figure 3.23.) Only a disk loss in the small reliability group will avoid data loss and instead render it vulnerable. Then any further disk loss amounts to data loss.

We can now give the probability that l additional disk failures in the original partition will lead to data loss. As we have seen, data loss will occur either with loss of the second or third disk. We calculate the probability for data loss with two additional disk failures by counting the number of having one disk fail in a different reliability group and the next one in the same reliability group as the first one. We formulate the answer in a theorem for easier reference:

Theorem 5 *The data failure probability of data loss given l additional failures in a partition with x reliability groups (and x failed disks) is*

$$\begin{aligned} p(x,l) &= 0 \qquad \textit{for } l = 0 \textit{ or } l = 1 \\ p(x,2) &= 1 - (x-1)n \cdot (n-1) \cdot \left(\begin{array}{c} x(n-1) \\ 2 \end{array} \right)^{-1} \\ p(x,l) &= 1 \qquad \textit{for } l \geq 3. \end{aligned}$$

The dependence on x in the above formula shows that the location of the first m disk failures is important. We illustrate the fact by giving in tabular form the data loss probabilities for two failed disk in a partition for our two running example arrays; the first one contains 5 reliability group with nine information disks, one check and one spare disk, the second one contains 10 reliability groups with ten information disks, one check and one spare disk.

(a)

x	$p_{dl}(x)$
1	1.000000
2	0.421053
3	0.494253
4	0.576923
5	0.640816

(b)

x	$p_{dl}(x)$
1	1.000000
2	0.428571
3	0.500000
4	0.581395
5	0.644444
6	0.692308
7	0.729323
8	0.758621
9	0.782313
10	0.801835

Figure 3.24: Data loss probability $p_{dl}(x)$ for the failure of two additional disks in a partition with x reliability group for the two example arrays: (a) The 5×11 array. (b) The 10×12 array.

(d) The Probability Space of the First m Disk Failures: The failure patterns form a probability space

$$A(m) = \{(x_1, x_2, \ldots, x_m) : \sum_{i=1}^{m} x_i = m\}$$

with probability measure

$$p(x_1, x_2, \ldots, x_m) = \prod_{i=1}^{m} \binom{n}{x_i} \binom{mn}{m}^{-1}$$

that gives the chances of this particular pattern occuring given that m disks have failed. In particular, x_i is observed with (hypergeometric) probability

$$p(x_i) = \binom{n}{x_i} \binom{(m-1)n}{m-x_i} \binom{mn}{m}^{-1}$$

(e) Recursive Calculation of Data loss Probabilities from Disk Failures Only: We can now give a recursive formula for the data loss probability given that $m+l$ disks have failed. We use recursion on numbers $F(s, w, l)$ where s stands for size and w for weight. The intuitive meaning of $F(s, w, l)$ is the relative data loss probability in reliability groups 1 to s when w of the first m disk failures and l of the additional disk failures occur in the first s reliability groups. Thus $F(m, m, l)$ denotes the data loss probability if additionally l disks have failed.

Our formula uses the probabilities $p(x,l)$ in Theorem 5 and the hypergeometric probabilities p of the preceding paragraph. We first distinguish cases according to x_s, the number of failed disk in reliability group s:

$$F(s,w,l) = \sum_{x_s=0}^{w} F(s,w,l|x_s)p(x_s|s,w).$$

Here $p(x_s|s,w)$ denotes the probability that x_s disks have failed in reliability group s, when we already know that a total of w have failed in the first s reliability groups:

$$p(x_s|s,w) = \left(\begin{array}{c} n \\ x_s \end{array}\right)\left(\begin{array}{c} (s-1)n \\ w-x_s \end{array}\right)\left(\begin{array}{c} sn \\ w \end{array}\right)^{-1}.$$

The conditional probabilities $F(s,w,l|x_s)$ are then calculated by distinguishing the number of additional disk failure in the partition if $x_s \neq 0$.

$$F(s,w,l|x_s) = \begin{cases} F(s-1,w,l) & \text{if } x_s = 0 \\ \sum_{\lambda=0}^{l} q_{x_s}(\lambda)\,(p(x_s,\lambda) + F(s-1,w-x_s,l-\lambda) & \\ \qquad -p(x_s,\lambda)F(s-1,w-x_s,l-\lambda)) & \text{if } 1 \leq x_s < w \\ p(s,l) & \text{if } x_s = w. \end{cases}$$

Here, $q_{x_s}(\lambda)$ denotes the probability that λ of the additional disk failures are located in the partition with leader s. This probability is hypergeometric:

$$q_{x_s}(\lambda) = \left(\begin{array}{c} x_s(n-1) \\ \lambda \end{array}\right)\left(\begin{array}{c} (w-x_s)(n-1) \\ l-\lambda \end{array}\right)\left(\begin{array}{c} w(n-1) \\ l \end{array}\right)^{-1}$$

The basis of the recursion is given by:

$$F(1,w,l) = p(w,l).$$

(f) From Absolute Data loss Probabilities to Transition Probabilities: We collapse the many disk failure states characterized by relevant failure patterns into states indexed only by the number of failed disks. As we can conclude from the previous discussions, only the $m + 2^{nd}$ and following disk failures can lead to data loss. Hence, the transition probabilities between the states describing l and $l+1$ disk failures is just the probability of a disk failure or $(mn-l)\lambda$, as long as $l < m+1$. For larger l, we have to take

the possibility that exactly the l^{th} disk failure leads to data loss into account. For $l > m+1$, the data loss probability $P_{dl}(l)$ that l disk failures lead to data loss, is $F(m, m, l)$ (which incidentally is 1 for $l \geq 2m$.) We denote the probabililty that exactly the l^{th} disk leads to data loss with $p_{dl}(l)$. If we distinguish according whether the $l-1$ disk failure caused data loss, we can write the probability for data loss for l disk failures as

$$P_{dl}(l) = P_{dl}(l-1) + p_{dl}(1 - P_{dl})$$

and hence

$$p(l) = \frac{P_{dl}(l) - P_{dl}(l-1)}{1 - P_{dl}(l-1)}.$$

Here l ranges between 1 and $2m$.

The marginal transition probability from State $l-1$ characterized by $l-1$ failed disk without data loss to State l is then

$$(1 - p_{dl}(l))(mn - m - l)\lambda.$$

We illustrate our results by giving the absolute failure probabilities and relative failure probabilities for our two running examples in Table 3.21.

(g) Consequences of String Failure: We cannot treat a string failure as failure of m disks, because the location of the previously failed disks matters and because previously failed disks can be situated on the failing string. In this section, we calculate the probability that a string failure does not lead to data loss after previous failure of l disks and that the resulting state of the RAID is characterized by string failure and additional k failed disks.

Our method consists in a careful case distinction of the partitioning after the first m disk failures. We first analyze the effects of string failure on a single partition. If the partition consists of one reliability group with one disk failure, the string failure cannot lead to local data loss. Depending on whether the failed disk is located on the failing string or not, this partition contributes zero or one to the count k of failed disks outside the failed string. If a one reliability group partition contains another failed disk, only failure of a string on which one of the two failed disks are situated will not lead to data loss. The contribution to k is one.

We examplify the more complicated situation, in which a partition consists of two reliability groups, in Table 3.25. Failure of a string on which the two already failed disks are

(a)

Number of Failed Disks	Abs. D.L. Prob. P_{dl}	Rel. D.L. Prob. p_{dl}
6	0.000000	0.000000
7	0.190771	0.190771
8	0.502666	0.385422
9	0.791667	0.581100
10	0.953247	0.775588

(b)

Number of Failed Disks	Abs. D.L. Prob. P_{dl}	Rel. D.L. Prob. p_{dl}
11	0.000000	0.000000
12	0.101188	0.101188
13	0.282369	0.201578
14	0.498656	0.301390
15	0.699554	0.400719
16	0.849641	0.499547
17	0.939516	0.597734
18	0.981549	0.694951
19	0.996135	0.790522
20	0.999547	0.882851

Table 3.21: Data Loss Probability for two disk arrays, (a) 5 reliability groups with 11 disks each and a total storage capacity equal to that of 45 individual disks and (b) 10 reliability groups with 12 disks each and a total storage capacity corresponding to that of 100 individual disks each.

not situated leads to data loss because the relocated data from these disks will sometimes be on the failed string. If the string contains one of the two disks, we avoid data loss. If in addition to the two failed disk another disk has failed in the partition, the only scenario that avoids data loss is: The additional disk lost is in the other reliability group (not the leader), and the failed string contains one of the two original disk failures. Depending on the location of the additional disk failures, one or two disk failures contribute to k. If two additional failures have occured, then the two additional are in the other reliability group and one of them as well as one of the original failed disks are on the failed string. If the partition started out with three reliability groups, then, regardless of additional disk failures, no string failure can be tolerated. We collect the exact probabilities of data survival in a partition of size x_s with l_s additional disk failures, so that in the resulting state k_s failed disks besides the string are in the original partition, in Table 3.22.

(a)

track 1:	a_1	a_2	a_3	
	b_1	b_2	b_3	
track 2:		a_1	a_2	a_3
		b_1	b_2	b_3
track 3:	a_3		a_1	a_2
	b_3		b_1	b_2

(b)

track 1:	a_1	a_2	■	■
	b_1	b_2	b_3	a_3
track 2:	a_2	a_1	■	■
	a_3	b_1	b_2	b_3
track 3:	a_3	a_1	■	■
	b_3	a_2	b_1	b_2

Figure 3.25: A Two by Four Disk Array: (a) Fault-Free Data Lay-Out (b) Data Lay-Out after 2 Disk Failures

x_s	l_s	k_s	P
1	0	0	$1/n$
1	0	1	$(n-1)/n$
1	1	1	$2/n$
2	0	1	$2/n$
2	1	1	$1/(n(n-1))$
2	1	2	$1/n$
2	2	2	$2/(n(2n-3))$

Table 3.22: Non-zero Probabilities, that in a partition with originally x_s reliability groups and l_s additional disk failures, the data survives string failure and that in the resulting state k_s disk failures outside the failed string remain.

(h) Transition Rates from Disk to String Failure States: We first calculate the transition rates from the state, in which exactly m disks have failed. We introduce the probability function $G(s, w, k)$ whose arguments have similar intuitive meanings as in Section (e). The first number, s gives the number of reliability groups we are considering, the second is the "weight", the number of disk failures in the s reliability groups and k is the number of disks failures outside the string in reliability groups 1 to s. From our analysis in Section (g) we derive the initial values:

$$\begin{aligned} G(1,1,0) &= \frac{1}{n} \\ G(1,1,1) &= 1 - \frac{1}{n} \\ G(1,2,1) &= \frac{2}{n} \\ G(1,0,0) &= 1 \end{aligned}$$

$$G(1,w,k) \;=\; 0 \qquad \text{for all other values.}$$

We obtain a recursion formula for $G(s,w,k)$ based on the partitioning:

$$\begin{aligned} G(s,w,k) \;=\;& p_{hp}(0,n,w,sn)G(s-1,w,k) \\ &+p_{hp}(1,n,w,sn)\frac{1}{n}G(s-1,w-1,k) \\ &+p_{hp}(1,n,w,sn)(1-\frac{1}{n})G(s-1,w-1,k-1) \\ &+p_{hp}(2,n,w,sn)\frac{2}{n}G(s-1,w-2,k-1) \end{aligned}$$

Here, p_{hp} stands for the hypergeometric probability distribution. In our sum, the first addend corresponds to the cases, in which the current reliability group is not a leader, the second to the case, in which the current reliability group is a leader with one lost disk and where the failed disk is located on the failing string.

If m disks have failed, then the probability that a string failure leads to a transition to the state characterized by a string failure and failure of an additional k disks is $G(m,m,k)$. The same probability under the assumption that only l (with $0 < l \leq m$) disks have failed, is $G(m,l,k)$.

Our final probability calculation determines the data loss probability for a string failure after previous $l+m$ disk failures. We define a data loss probability function $K(s,w,l,k)$ recursively, where the arguments have the same intuitive meaning as before and l denotes the number of additional failures located in reliability groups with leader in groups 1 to s.

The recursion base is laid in Table 3.22. The recursion itself takes the form

$$\begin{aligned} &K(s,w,l,k) = \\ &p_{hg}(0,n,w,sn)K(s-1,w,l,k) \\ &+p_{hg}(1,n,w,sn)p_{hg}(0,n-1,l,wn)\tfrac{1}{n}K(s-1,w-1,l,k) \\ &+p_{hg}(1,n,w,sn)p_{hg}(0,n-1,l,wn)\tfrac{n-1}{n}K(s-1,w-1,l,k-1) \\ &+p_{hg}(1,n,w,sn)p_{hg}(1,n-1,l,wn)\tfrac{2}{n}K(s-1,w-1,l-1,k-1) \\ &+p_{hg}(2,n,w,sn)p_{hg}(0,2n-2,l,wn)\tfrac{2}{n}K(s-1,w-2,l,k-1) \\ &+p_{hg}(2,n,w,sn)p_{hg}(1,2n-2,l,wn)\tfrac{1}{n(n-1)}K(s-1,w-2,l-1,k-1) \\ &+p_{hg}(2,n,w,sn)p_{hg}(1,2n-2,l,wn)\tfrac{1}{n}K(s-1,w-2,l-1,k-2) \\ &+p_{hg}(2,n,w,sn)p_{hg}(2,2n-2,l,wn)\tfrac{2}{n(2n-3)}K(s-1,w-2,l-2,k-2) \end{aligned}$$

The various summands arise from case distinctions, where we first separate according to whether reliability group s is not or is a leader (summand 1 versus all the other ones) and whether 1 or 2 disks have failed, then according to the number of additional disk failures in each partition. The first two factors in a summand are the respective probabilities, then we multiply the survival probability in the partition with a certain number of surviving disk with the corresponding survival probability in the rest of the disk array. We need not worry about partitions with leaders with weight 3 or more, as those always suffer data loss during a string failure.

(i) Disk Failures after String Failures: After a string failure, every reliability group suffered loss of at least one disk and hence every reliability group is vulnerable. Our reassignment policy insures, that the spare space in each reliability group is devoted to lost data in the same group. Hence, the situation after a string failure is completely characterized by the number of additional disk failures. The careful reader will not need the caveat, that this number of additional disk failures is not the total number of historic disk failures, as some of these might have been located on the failed string. The only disk failures that do not cause data loss are in a reliability group without additional disk failures, hence the marginal probability for a non-data loss transition is $(m-k)n\lambda$ for k additional disks.

(j) Second String Failure: A second string failure leads to data loss exactly if any additional disk failures are not located on the second failed string. If this number is k, then the data loss probability is $1-n^{-k}$. After the second string failure, the system either has already suffered data loss or is not capable of withstanding any further component losses.

(k) The Markov Model: We distinguish disk failure states (D,l), where l, with $0 \leq l \leq 2m$, disks have failed, string failure states (S,k), with $(0 \leq k \leq m)$, which describe a string failure and an additional k disk failures. Finally, we have the double string failure state $(2S)$. We present the probability that a given transition is taken for our RAID with 5 reliability groups in Table 3.23.

(D,l)	to $(D,l+1)$ to $(S,3)$	to $(S,0)$ to $(S,4)$	to $(S,1)$ to $(S,5)$	to $(S,2)$
0	1.000000 0.000000	1.000000 0.000000	0.000000 0.000000	0.000000
1	1.000000 0.000000	0.090909 0.000000	0.909091 0.000000	0.000000
2	1.000000 0.000000	0.006734 0.000000	0.168350 0.000000	0.673401
3	1.000000 0.381170	0.000381 0.000000	0.019059 0.000000	0.190585
4	1.000000 0.146604	0.000015 0.146604	0.001466 0.000000	0.029321
5	1.000000 0.028746	0.000000 0.071865	0.000072 0.028746	0.002875
6	0.809229 0.001533	0.000000 0.008681	0.000001 0.011743	0.000080
7	0.614578 0.000074	0.000000 0.000779	0.000000 0.002074	0.000002
8	0.418900 0.000002	0.000000 0.000054	0.000000 0.000250	0.000000
9	0.224412 0.000000	0.000000 0.000002	0.000000 0.000026	0.000000
10	0.000000 0.000000	0.000000 0.000000	0.000000 0.000003	0.000000

Table 3.23: Transition Probabilities for the 5×11 classic RAID with Distributed Sparing.

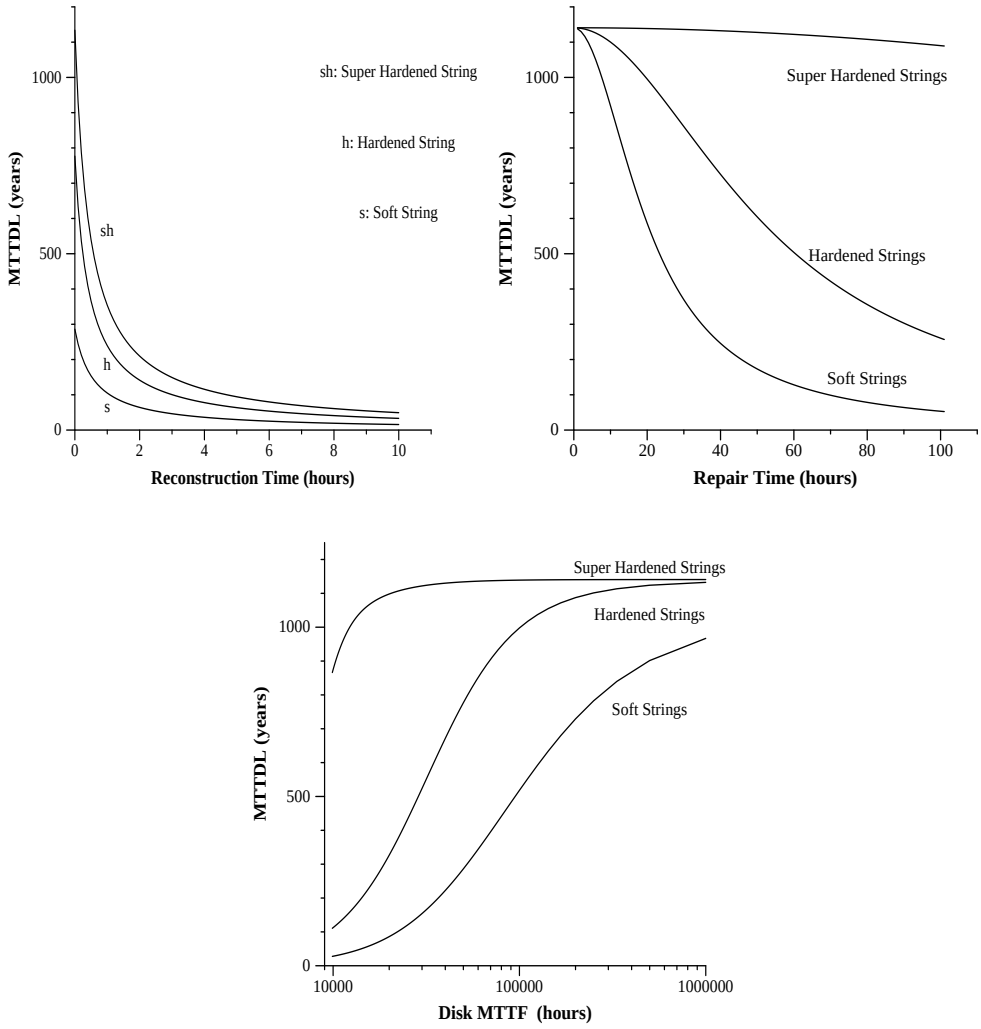


Figure 3.26: Impact of Reconstruction Time, Repair Time, Disk MTTF on the MTTDL of the Classic RAID with Distributed String Sparing.

3.4.9 Distributed String Sparing with Almost Complete Load Balancing

This organization combines excellent reliability for all string hardness levels with excellent performance in the normal and in the disk failure case.

To make the analysis of the Markov model numerically safer, we again collapse states. We characterize the states of our Markov model by the number of failed disks with or without the conjunction of one or two string failures. The analysis is more involved than the pure accounting in a model with many states, but our error estimates are still excellent.

First, we determine the number of failed disks that the RAID can occur without data loss. The first m disk failures are reconstructed on the spare space. The next disk failure stresses the redundancy of the RAID. Disk failure $m+2$ deserves careful analysis. If it occurs on a different string than disk failure $m+1$, data loss is assured. If disk failures $m+1$ and $m+2$ are located on the same string, data loss happens, if these two disks, which failed last, contain data from the same reliability group that got there through a reconstruction process. This cannot happen, if the first m failures were located on the same string. Otherwise, the probability of data loss is very high. For example, if only two disks are located on different strings, it is $1-10^{-20}$ for the 5×11 RAID and $1-10^{-8}$ for the 10×12 array. We conclude that the $m+2$ disk failure does lead to data loss, unless the first m disk failures occur on the same string and the next two disk failures are located on the same string, too.

We distinguish disk failure states (D,l) with l ranging from 0 to $m+1$ in our analysis. Here l is the number of failed disks. We have states (S,l), with $0 \leq l \leq m$ that describe string failure in conjunction with l disk failures, that do not lead to data loss. In state (S,l) the l additional disk failures are located on the same string, therefore, state (S,m) describes two string failures as well. We describe failure of m disks on the same string by transition to state $(S,0)$. For this we adjust the disk failure transition rate from $(D,m-1)$ to either (D,m) and to $(S,0)$.

We now give the state transition rates, first for disk failures. We have a state transition from (D,l) to $(D,l+1)$ at a rate of $(nm-l)\lambda$. However, if $l=m-1$, we need to adjust for our state relabeling. We observe a transition to $(S,0)$ when all m failed disks

are located on the same string. The transition rate is therefore

$$n \cdot \begin{pmatrix} nm \\ m \end{pmatrix}^{-1} (mn - m + 1)\lambda$$

and to (D, m) it is

$$(1 - n \cdot \begin{pmatrix} nm \\ m \end{pmatrix}^{-1})(mn - m + 1)\lambda$$

To discuss the impact of a string failure on a disk array with l previous disk failures, we notice first that if - on a given track level - the string was not used as spare space and $l \leq m$, no dataloss can occur. The situation is different for a string that is used as spare space. This situation occurs approximately t/n times, where t is the number of tracks. k of the l failures are located on the failed string with hypergeometrical probability $p_{hg}(l-k, n, l, nm)$. If the string serves as spare space, then dataloss occurres if two (or more) of the replaced data belonged to the same reliability group. This can only happen if the failed disks are located on different strings. If there are only two failed disks on different strings, the data survival probability is $(1-1/m)^{t/n}$ or $1.9*10^{-9}$ and $1.77*10^{-4}$ for our two examples. If there are three failed disks on two different strings, then the data survival rate sinks to $(1-2/m)^{t/n}$ or $1.08*10^{-20}$ and $1.77*10^{-8}$ for our two examples. These numbers justify to disregard the possibility for data survival in these cases.

As an aside, our calculations assume a random assignment of reliability groups to disks in a string. In a concrete disk array, this assignment is fixed. Our calculations signify that it is difficult to find an addressing scheme that avoids dataloss in these cases. Also, performance after a disk loss is maximized by an addressing scheme that would guarantee dataloss in these cases.

We can now gather the string failure transition probabilities for the various disk failure states:

From Normal State $(D, 0)$ a string failure leads with marginal probability $n\mu$ to State $(S, 0)$.

From State $(D, 1)$ a string failure leads with marginal probability μ to State $(S, 0)$ and with marginal probability $(n-1)\mu$ to State $(S, 1)$.

From State $(D, 2)$ a string failure leads with marginal probability $\frac{n(n-1)\mu}{m(nm-1)}$ to State $(S, 0)$,

with marginal probability $\frac{2n^2(m-1)}{m(nm-1)}$ to State $(S,1)$ and with marginal probability

$$\left(\binom{n-1}{2}n^2\binom{nm}{2}^{-1}+(n-1)\binom{n}{2}\binom{nm}{2}^{-1}(1-1/m)^{t/n}\right)n\mu$$

to State $(S,2)$.

In general, if the system is in State (D,l), then k disks will be located outside the string with hypergeometric probability

$$p_{hg}(l-k,n,l,nm)=\binom{n}{l-k}\binom{mn-n}{k}\binom{nm}{l}^{-1}$$

are located outside the failing string. If $k=2$, then with probability

$$\left(\binom{n-1}{2}n^2\binom{m(n-1)}{2}\cdot(2/m)\right)^{t/n}$$

we observe dataloss, here, the first factor is the chance that the two disks are on different strings and the second factor the chance of dataloss in that case. With the opposite probability we observe a transition to State (D,k). A string failure itself occurs with marginal probability $n\mu$.

If $k\geq 2$ then with probability

$$(n-1)\binom{m}{k}\binom{m(n-1)}{k}^{-1}$$

the disks are all located on the same string and we observe a transition to State (S,k). Otherwise, we observe dataloss.

Next, we consider disk failures after a string failure. If we are in State $(S,0)$ we observe a transition to $(S,1)$ with marginal probability $(n-1)m\lambda$. In State (S,l) with $l\geq 1$, any additional disk failure in a different string will lead to dataloss, this transition represents with marginal probability $(n-2)m\lambda$, and one in the same string as the other additional disk failures will keep the data safe. The transition to State $(S,l+1)$ occurs with probability $(n-l)\lambda$.

A string failure from State $(S,0)$ happens with probability $(n-1)\mu$ and leads to a State $(2S)$ from which any other failure leads to dataloss (with marginal probability

$(n-2)m\lambda + (n-2)\mu$.) From State (S, l) dataloss is only avoided if all l disks are located on the failing string: With probability

$$(n-1)\begin{pmatrix} m \\ l \end{pmatrix}\begin{pmatrix} (n-1)m \\ l \end{pmatrix}^{-1}\mu$$

we observe a transition to State $(2S)$ and with probability

$$(n-1)\left(1-\begin{pmatrix} m \\ l \end{pmatrix}\begin{pmatrix} (n-1)m \\ l \end{pmatrix}^{-1}\right)\mu$$

a transition to the failure state.

We give the sensitivity analysis in Figure 3.27. It becomes clear from the graph for the reconstruction time, that the excellent resilience depends on fast reconstruction. This graph and its companion graph in the preceding section constitute a strong argument for ACATS.

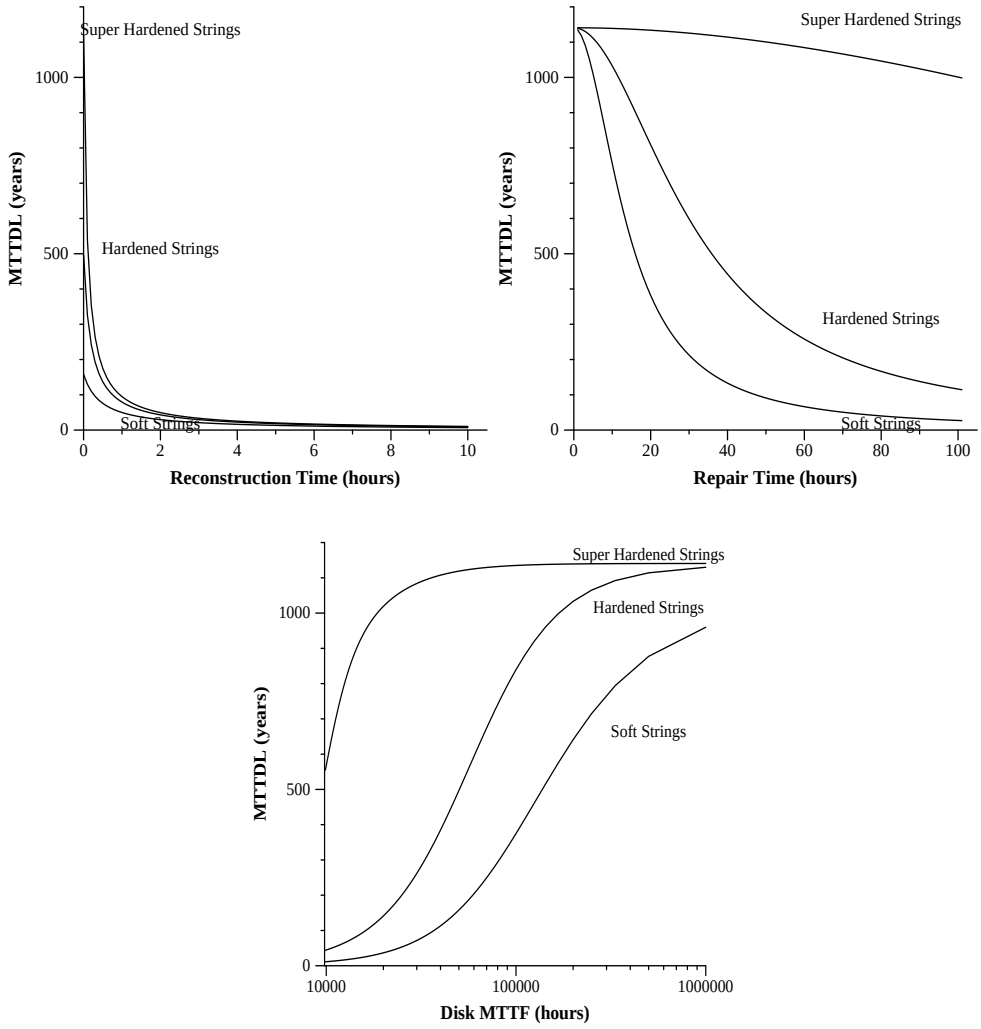


Figure 3.27: Impact of Reconstruction Time, Repair Time, Disk MTTF on the MTTDL of the RAID with ACATS and Distributed String Sparing.

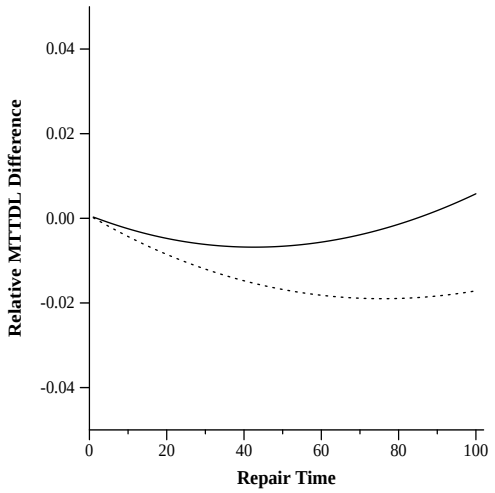


Figure 3.28: Relative Difference of MTTDL obtained from the Deterministic Repair Time Distribution (solid line) and Uniform Repair Time Distribution (dotted line) compared with MTTDL obtained using the exponential distribution.

3.5 Influence of Repair Time Distribution

We investigate the influence of the repair time distribution on RAID MTTDL in this section. We consider the example of the Level 5 RAID with ACATS and no sparing. We use two alternative repair time distributions. The first is the deterministic distribution, where repair takes place exactly $1/\rho$ hours after a failure. The uniform distribution sees repairs taking place with equal probability during a fixed amount of space. A "realistic" distribution is a mixture of these two. For illustration, we assume that failures are discovered almost immediately during an 8 our daily shift. Repair then takes place within a certain time limit (e.g. 1 hour). A failure at other times is discovered at the beginning of a shift. The resulting, realistic probability distribution for the repair time has a repair time of 1 hour with probability 1/3 and distributes the remaining 2/3 weight uniformally between 1 hour and 9 hour repair times (assuming dedicated employees willing to tackle problems on overtime.)

To calculate the MTTDL under these distributions, we first calculate the probability that after a disk or string failure another failure happens before the repair has been

performed. In case of the deterministic distribution, this probability is $p^{(1/\rho)}$ where p denotes the hourly rate of further dataloss inducing failure and $1/\rho$ the time between the initial failure and repair. If the repair time is uniformally distributed between 0 and $2/\rho$, we obtain the probability of further failure as

$$\frac{\rho}{2\ln(p)} \cdot \left(p^{2/\rho} - 1\right).$$

We calculate the MTTDL now from a two state failure model. A transition takes place from the fault free state to the data loss state if an initial failure happens and another failure strikes while the repair triggered by the first failure is not performed. The resulting MTTDL figure does not account for time the system spends waiting for repair, but stops the clock ticking until the repair has been performed. We adjust the preliminary MTTDL figure by adding the average repair time $1/\rho$ times the expected number of repair triggering failures.

We compare the resulting MTTDL figures with the ones for the exponential distribution and display the results in Figure 3.28, where we give the relative difference. The exponential distribution gives the largest MTTDL numbers, followed by the uniform and the deterministic repair time distributions. The error resulting from use of the wrong distribution is less than 2 % . Garth Gibson [11] comes to the same conclusion as we in his Ph. D. thesis: The impact of repair time distribution is minimal.

Chapter 4

MDS Based RAIDs

4.1 MDS Codes

In this section we give only a short overview of the coding theoretical background of Maximum Distance Separable (MDS) codes. A more complete treatment of the topic can be found in any good book on algebraic coding theory [26].

4.1.1 Holographic Information Dispersal

In [30] M. Rabin introduced the Information Dispersal Algorithm (IDA.) IDA takes a given datum and generates m equally sized fragments. Given n or more fragments, (whose number depends on the IDA algorithm used) we can reconstruct the original datum. Figure 4.1 illustrates an IDS algorithm with 8 fragments and a threshold of 4. This holographic storage algorithm can be formulated in terms of coding theory and the considerable knowledge about algebraic codes can be applied. [29] was among the first to point out the connection between Rabin's approach and MDS Codes. This should not distract from Rabin's achievement who gave a new paradigm for well known mathematical tools. Karnin, Greene, and Hellman[15] who studied the sharing of information within a "faulty environment" use a similar scheme. Abdel-Ghaffar and El Abbadi report a file comparison scheme that uses this approach [1]. Recently Blaum *et al.* [4] have created a novel coding scheme for the explicit purpose of use in a disk array with double disk failure tolerance.

To make the connection between IDA and coding theory, assume that a bit string I is subjected to IDA, resulting in fragments $c_1, \ldots c_m$. Then the mapping of I to the

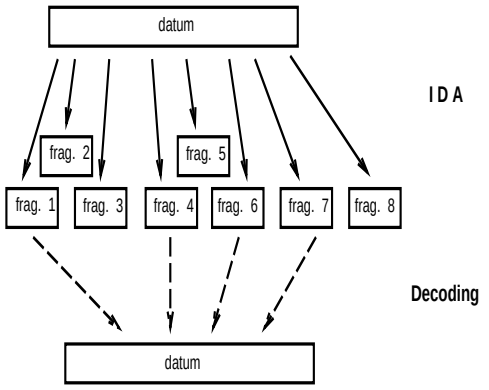


Figure 4.1: General IDA Scheme

concatenation of all the fragments $c_1c_2 \ldots c_m$ is an algebraic block code. We assumed that all the fragments lie in a fixed symbol set; to be more precise, they are represented as bit strings of a given length. By design an IDA algorithm yields an $m - n$ erasure correcting code: if $m-n$ of the symbols in a codeword are lost, then the codeword can still be correctly decoded. Erasure correction is easier than error correction, indeed most of the work in error correcting decoding is spent on calculating the error location. Every error correcting code is erasure correcting at the same or - usually at a much higher - level of correction.

Besides ease of computation, an efficient IDA is one that makes good use of the space used to store the fragments. A *storage optimal* IDA uses as much storage (in bits) for n fragments as the original data uses. We cannot hope to do better, unless the original storage scheme for the data is redundant. A space optimal IDA has to generate fragments of the same length L/n, if L is the size of the original datum. While the possible data lengths varies, an IDA algorithm has to assume that the data consists of atomic constituents, the data symbols. For a practical algorithm, a data symbol could consists of a bit strings of length 16 bits and disperse the symbols into fragments composed of symbols of length 8 bits each. While not every possible bit string can be exactly decomposed in symbols of 16 bits each, we can pad with zeroes.

The corresponding encoding is a mapping from the data symbol space (all bits of length l) into the m – fold cartesian product of the fragment symbol space (m-fold concatenation of bit strings of length $k = l/n$.) The size of the data symbols has to be a

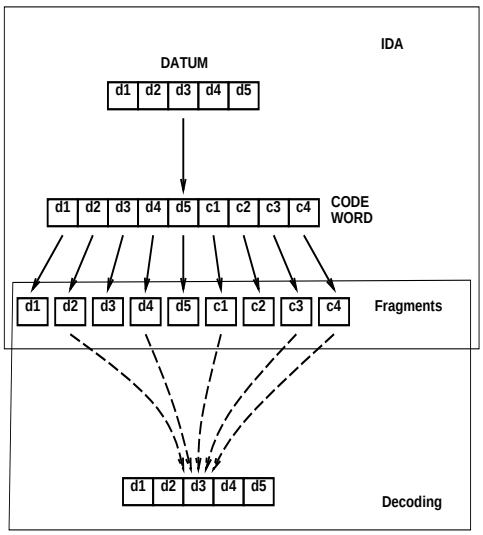


Figure 4.2: Wastefree IDA as an MDS code.

multiple of the fragment symbols. In all, the encoding is a mapping from the $n$fold cartesian space over the fragment symbols into the $m$fold cartesian space, or an [n,m] code. (The codewords (the concatenation of the fragments) form an n dimensional manifold in an m dimensional linear space over the finite field with 2^k elements, where k is the fragment lenght in bits.)

The IDA property states that a codeword is determined if n of the codeword coordinates (the fragments) are known. Codes with this property are known as *Maximum Distance Separable Codes* (MDS) codes. The name refers to an equivalent property of codes. We can easily change an MDS codes to separate information symbols from check symbols. Then a codeword is formed from a datum $(d_1, d_2, \ldots, d_n)$ by appending check symbols $c_{n+1}, \ldots c_m$ to obtain a code word

$$(d_1, d_2, \ldots d_n, c_{n+1}, c_{n+2}, \ldots c_m).$$

The resulting IDAs are by far better than the original examples given by Rabin. We illustrate the connection in Figure 4.2.

4.1.2 Linear MDS Codes

The best suited MDS codes for our purposes form a certain class of linear MDS codes that happens to be already in wide use for error correcting data transmissions and is implemented in hardware on a commercially available chip. As we will never use the error correction capabilities of the codes, special purpose chips would be smaller and even faster. In fact, the speed of the chips available is more than sufficient for use in a disk array.

The description of this family of codes relies on linear algebra. In short, a datum is represented as a vector of dimension n over the finite field with 2^k elements. We then multiply this vector with a $m \times n$ matrix T to obtain the codeword as an m-dimensional vector. The matrix T is chosen so that any n rows are linearly independent. Vandermonde's theorem assures us that such matrices exist with 2^k rows. This is more than sufficient, if we use $k \geq 4$ or $k = 8$ and base our scheme on bytes. In order to decode a codeword, we only need to know n of its vector coordinates. We form a matrix $\hat{T}$ consisting of only those rows in T, for which we know the corresponding coordinates in the codeword. Then $\hat{T}$ is invertible and we can recover the original datum by solving a system of linear equations.

By using elementary column transformations and elementary row transformations for the first n rows only, we can always transform a Vandermonde matrix into an encoding matrix T which has the identity matrix in the first n rows. As a consequence of this normalization, the codeword formed from datum $(i_1 i_2 \ldots i_n)$ has the same symbols in the first n coordinates. In a formula:

$$\begin{pmatrix} i_1 \\ i_2 \\ \vdots \\ i_n \\ c_{n+1} \\ \vdots \\ c_m \end{pmatrix} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ t_{n,1} & t_{n,2} & \cdots & t_{n,n} \\ t_{n+1,1} & t_{n+1,2} & \cdots & t_{n+1,n} \\ \vdots & & \ddots & \vdots \\ t_{m,1} & t_{m,2} & \cdots & t_{m,n} \end{pmatrix} \cdot \begin{pmatrix} i_1 \\ i_2 \\ \vdots \\ i_n \end{pmatrix}$$

Because the encoding is linear we have a remarkable and extremely useful property. If one coordinate of the datum is changed, then only the check symbols and the corresponding information coordinate in the codeword changes. Furthermore, the new check symbols

+	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

*	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	3	1
3	0	3	1	2

Table 4.1: Finite Field Operations for the Field $\{0, 1, 2, 3\}$

can be calculated solely from the change in the information coordinate and the old check symbol. The calculation involves only multiplication and addition.

4.1.3 A Coding Example

We give an example for linear MDS codes. To simplify notation we use 2-bit symbols, which we denote by the decimal equivalence of the binary number they represent. Thus 0 denotes the bitpattern 00 and 2 the bitpattern 10. All our calculations are done in the finite field with 2 elements. We assume the standard definition of addition as the exclusive-or of bit strings and multiplication as polynomial multiplication modulo the one irreducible polynomial over $\{0, 1\}$. We give the addition and multiplication table in Table 4.1.2.

The order 3 Vandermonde matrix has rows consisting of the powers of a given element x, that is, the rows have the form (x^0, x^1, x^2). In addition, we can add an additional row $(0, 0, 1)$ to obtain the "once extended" VanderMonde matrix:

$$V = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 3 & 2 \\ 0 & 0 & 1 \end{pmatrix}$$

We use now elementary transformations to obtain our Coding Matrix

$$C = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 2 & 3 \end{pmatrix}$$

The coding matrix is not unique, our particular choice offers easy arithmetic.

Assume that we would want to encode a bit string

$$00100001001111100100001000\ldots.$$

We then split the bit string into blocks of 6 bits each:

$$001000 \quad 010011 \quad 111001 \quad 000010 \quad \ldots,$$

organize the substrings as three dimensional vectors over our finite field:

$$\begin{pmatrix} 0 \\ 2 \\ 0 \end{pmatrix} \quad \begin{pmatrix} 1 \\ 0 \\ 3 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 0 \\ 2 \end{pmatrix} \quad \ldots$$

Then we multiply these vectors with our coding matrix C and obtain:

$$\begin{pmatrix} 0 \\ 2 \\ 0 \\ 2 \\ 3 \end{pmatrix} \quad \begin{pmatrix} 1 \\ 0 \\ 3 \\ 2 \\ 3 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 0 \\ 2 \\ 2 \\ 1 \end{pmatrix} \quad \ldots.$$

We convert the vectors to 10 bit strings:

$$0010001011 \quad 0100111011 \quad 0000101001 \quad \ldots$$

and concatenate to obtain the encoding of our original bit string

$$00100010110100111011000010100 1\ldots.$$

This is the bit-version of our MDS code used as an error correcting code.

If we wanted to use our encoding mechanism for IDA, we would proceed slightly different. The first steps are the same, we split the original sequence into strings of 6 bits each

$$001000 \quad 010011 \quad 111001 \quad 000010,$$

write them as three dimensional vectors

$$\begin{pmatrix} 0 \\ 2 \\ 0 \end{pmatrix} \quad \begin{pmatrix} 1 \\ 0 \\ 3 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 0 \\ 2 \end{pmatrix} \quad \dots$$

and encode them by multiplication from the left with C:

$$\begin{pmatrix} 0 \\ 2 \\ 0 \\ 2 \\ 3 \end{pmatrix} \quad \begin{pmatrix} 1 \\ 0 \\ 3 \\ 2 \\ 3 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 0 \\ 2 \\ 2 \\ 1 \end{pmatrix} \quad \dots$$

The fragments are then formed by the coordinates in the same dimension, that is

$$(0,1,0,\dots) \quad (2,0,0\dots) \quad (0,3,2,\dots) \quad (2,2,2,\dots) \quad (3,3,1\dots)$$

or - as bit strings -

$$(000101\dots) \quad (100000\dots) \quad (001110\dots) \quad (101010\dots) \quad (111101\dots)$$

To show the encoding process, assume that fragments 3,4,5 in the above order are available. This correspond to a small encoding matrix c formed from rows 3,4 and 5 of C.

$$c = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 2 & 3 \end{pmatrix}$$

The inverse of c is

$$c^{-1} = \begin{pmatrix} 2 & 3 & 2 \\ 3 & 2 & 2 \\ 1 & 0 & 0 \end{pmatrix}$$

We organize the three available fragments into three-dimensional vectors, whose coordinates are taken from the corresponding fragment:

$$\begin{pmatrix} 0 \\ 2 \\ 3 \end{pmatrix} \quad \begin{pmatrix} 3 \\ 2 \\ 3 \end{pmatrix} \quad \begin{pmatrix} 2 \\ 2 \\ 1 \end{pmatrix} \quad \cdots$$

and multiply c^{-1} from the left to each of the vectors. The result is the vector sequence

$$\begin{pmatrix} 0 \\ 2 \\ 0 \end{pmatrix} \quad \begin{pmatrix} 1 \\ 0 \\ 3 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 0 \\ 2 \end{pmatrix} \quad \cdots$$

back, which unravels to

$$001000 \qquad 010011 \qquad 111001 \quad 000010,$$

or the original bitstring

$$0010000101101001110110000010100\ldots.$$

For MDS-code based RAIDs, we will use a slightly different scheme, that corresponds to taking the parity of unrelated data. Assume that we want to store the beginning of the Gospel of John in English, German and Greek. (In the beginning .., Im Anfang .., En en arche ..) In ASCII, the strings would start out with:

01001001011011100010000001110100

01001001011011010010000001000001

01000101011011100010000001100101

We convert the bit string into our 2 bit equivalent symbols and obtain

1021123202001310

1021123102001001

1011121201001211

As a vector sequence, this is

$$\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 2 \\ 2 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 2 \\ 2 \\ 2 \end{pmatrix}, \begin{pmatrix} 3 \\ 3 \\ 1 \end{pmatrix}, \ldots$$

We multiply with C in order to obtain our five fragments. Because of the form of C, the first three coordinates of each vector will just be the three-dimensional vector:

$$\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 2 \\ 2 \\ 1 \\ 1 \\ 2 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 2 \\ 2 \\ 2 \\ 2 \\ 0 \end{pmatrix}, \begin{pmatrix} 3 \\ 3 \\ 1 \\ 1 \\ 3 \end{pmatrix}, \ldots$$

We store then the original sequences on the first three disks and the two checks on two additional disks:

Disk 1: 1021123202001310

Disk 2: 1021123102001001

Disk 3: 1011121201001211

Disk 4: 1020000101001100

Disk 5: 0020003102000221

If, for example, disk 1 and 2 fail, then we can still recover the original data strings from disks 3, 4 and 5.

Assume now, that the first record ("In the beginning ...)" is overwritten by "In initio ...". The new data is encoded as

$$0100100101101110001000000101100 1\ldots$$

or (in our notation) by

$$1021123202001121\ldots.$$

Instead of recalculating the check sums by accessing all "information" disks 1,2 and 3, we use an alternative approach. The current checks are calculated by multiplying C to the information vectors:

$$\begin{pmatrix} i_1 \\ i_2 \\ i_3 \\ c_1 \\ c_2 \end{pmatrix} = C \cdot \begin{pmatrix} i_1 \\ i_2 \\ i_3 \end{pmatrix}$$

Now we are changing i_1 to i'_1.

$$\begin{pmatrix} i'_1 \\ i_2 \\ i_3 \\ c'_1 \\ c'_2 \end{pmatrix} = C \cdot \begin{pmatrix} i'_1 \\ i_2 \\ i_3 \end{pmatrix}$$

As the addition is really the exclusive-or, we calculate

$$\begin{aligned} \begin{pmatrix} i'_1 \\ i_2 \\ i_3 \\ c'_1 \\ c'_2 \end{pmatrix} &= \begin{pmatrix} i_1 \\ i_2 \\ i_3 \\ c_1 \\ c_2 \end{pmatrix} + \left(\begin{pmatrix} i'_1 \\ i_2 \\ i_3 \\ c'_1 \\ c'_2 \end{pmatrix} + \begin{pmatrix} i_1 \\ i_2 \\ i_3 \\ c_1 \\ c_2 \end{pmatrix} \right) \\ &= C \cdot \begin{pmatrix} i_1 \\ i_2 \\ i_3 \end{pmatrix} + \left(C \cdot \begin{pmatrix} i'_1 \\ i_2 \\ i_3 \end{pmatrix} + C \cdot \begin{pmatrix} i_1 \\ i_2 \\ i_3 \end{pmatrix} \right) \\ &= C \cdot \begin{pmatrix} i_1 \\ i_2 \\ i_3 \end{pmatrix} + C \cdot \begin{pmatrix} i'_1 + i_1 \\ 0 \\ 0 \end{pmatrix} \end{aligned}$$

For the checks in particular, this equation means

$$\begin{aligned} c'_1 &= c_1 + C_{4,1} \cdot (i'_1 + i_1) \\ c'_2 &= c_2 + C_{5,1} \cdot (i'_1 + i_1) \end{aligned}$$

where $C_{4,1}$ is the matrix entry in the fourth row and first column of C. Hence, we calculate the new check from the information Δ, the difference between the new and the old information, and the old check. The difference to parity coding lays in the multiplication.

For our update, first form the delta-value or difference between the new and the old string:

$$\Delta_1 = 0000000100000231\ldots$$

and multiply with one. Then we add the result to the old check information and overwrite the old checks on disks 4 and 5.

$$c'_1 = 1020000101001100 + 0000000100000231 = 1020000100001331$$

$$c_2' = 0020003102000221 + 0000000100000231 = 0020003202000010$$

4.2 Balanced Information Dispersal Algorithm

Burkhard, Claffy and Schwarz [7] investigate an application of IDA for fast disk based storage systems, the Balanced Information Dispersal Algorithm (BIDA.) By using an IDA algorithm by applying the database ideas of read/write quora and version numbers, we can offer a wide spectrum of excellent data safety as well as fast read or fast write times. A drawback of this scheme is the large number of disks used.

BIDA uses an array of independent disks as the storage device. An incoming block is dispersed using an IDA scheme. The fragments are all written to different disks, but the operation is successful if a write quorum of disks have been written. A read is directed to all disks, but has completed if fragments from a read quorum of disks have been recovered. An up-to-date version number validates fragments that have been actually written during the update of the block.

The scheme will do well in an environment with many small random accesses to storage. Most of the time, far more than the write quorum of disks will have been actually written, thus making reads normally very fast, while a write burst can be handled by only writing to the minimum number of disks. The organization stores data securely because more than the threshold number of disks will have been written in order to allow faster reads.

In the final analysis, BIDA allows to selectively improve access times beyond the single device technology. The throughput of BIDA (which is not that much better than that of a single disk) and the number of disks involved (10 to 20 per single disk capacity) do not make it attractive in most general purpose computing environments. Its storage costs compare favorably with electronic disks (DRAM). We can superimpose a BIDA scheme on a RAID to prevent a temporary storage overflow, which in effect builds a disk based non-volatile cache: A small part of the disk area is set apart for this purpose. If a write burst threatens to overflow the RAID cache (DRAM), then part of its contents are stored provisionally with BIDA. While storage costs might be as much as 20 times as high as storage using the RAID scheme, only a minute part of the total capacity needs to be used.

On the other hand, writes can be easily four times as fast as a RAID write (in its fastest implementation.)

A similar application of IDA to RAIDs has been proposed by Bestavros [3]. Bestavros analyzes what amounts to a full write quorum in our notation.

4.3 MDS RAIDs

We can use MDS codes to provide higher redundancy than Level 5 RAIDs. Instead of only keeping the parity of the data on all the disks in a reliability group on a separate disk, we can store two or three checks of an MDS codeword on separate disks. This allows us to compensate for the failure of two or three, respectively, disks in the group. The mechanics of storing are very similar to that of a parity based RAID. A write to an information block first reads the old block contents. On the second pass, the new block contents are written. In the meantime, the delta between the information block contents is determined. In parallel, the check blocks are read and when the delta value is available are overwritten with the new check blocks, calculated from the delta and the old check block contents.

Because the actual calculation is done very fast on chip, the only important aspect in which the write operation differs from the one at a parity based RAID is the higher number of check disks.

4.4 Reliability of some MDS RAID Organizations

MDS RAIDs display resilience against simultaneous unit failures to a much higher degree than the corresponding Level 5 RAID organizations with a full distributed string of spares. However their reliability is worse against unrelated component failure. We will give in Section 4.5 a simple scheme that gives an MDS RAID exactly the same reliability as for the Level 5 RAIDs with a distributed spare string. As the hardware costs are equal, the only trade-off between the two types consist in lower disk utilization for the same load evidenced by the Level 5 RAIDs and the better resilience against simultaneous multiple failures.

We only give the reliability of MDS RAIDs without reconfiguration. We will not pursue analysis of MDS RAIDs with extensive sparing. The result are given in Table 4.2.

Disk Array Organization	Storage Capacity (in disks)	MTTDL (in years) Super Strings	MTTDL (in years) Hard Strings	MTTDL (in years) Soft Strings
MDS (ACATS)	45	90.34	79.44	55.29
MDS (ADATS,1SDS)	44	820.82	760.07	491.39
MDS (ACATS)	100	10.31	9.65	7.95
MDS (ADATS,1SDS)	99	127.78	116.91	84.94
MDS (Classic)	45	754.92	580.66	259.48
MDS (Classic,1SDS)	44	983.36	940.51	613.09
MDS (Classic)	100	483.38	315.96	106.46
MDS (Classic,1SDS)	99	919.82	789.50	335.55

Table 4.2: MTTDL Values

The component reliabilities are those given in Table 2.1.

4.4.1 MDS Extension of the Classic RAID

The scheme organizes the disks in reliability groups with two check disks per group. Address Translation is only provided within the reliability group. The organization tolerates loss of two strings, loss of a string and at most one disk per group or loss of at most two disks per group. The reliability is superior, the performance in the normal state (no component failure) good (with use of volatile storage) and mediocre immediately after a disk failure. The comparison with the classic RAID with distributed (string) sparing shows, that the latter scheme gives better reliability numbers, if the reconstruction time is less than an hour. This does not include related component failures which might be prompted by the larger load of reconstruction or - more likely - by the same cause that contributed to the first component failure. A detailed analysis is given in Section 4.6.1.

4.4.2 MDS RAID with Almost Complete Address Translation

This scheme trades reliability for better performance after a disk failure. Its only difference to the Level 5 RAID described in 3.3.3 is the inclusion of another check disk in each reliability group. The RAID suffers data loss exactly if data on three disks in different strings become unavailable. Comparison with distributed sparing of a string shows that the latter scheme has better reliability for reconstruction times less than 1.5 hours.

4.4.3 MDS RAIDs with Safe Distributed Sparing

By adding one distributed spare disk to the scheme, we achieve reliability figures which are better than any other scheme over a wide range of string failure possibilities. Our sensitivity analysis in Figure 4.7 shows that the resulting reliability does not heavily depend on the reconstruction time.

4.5 Reconfiguration in MDS RAIDs

Two facts explain the better reliability of the corresponding Level 5 RAIDs with a full distributed spare string as compared to MDS RAIDs without sparing. While the MDS RAID does not need reconstruction on spare, the speed of the reconstruction of lost data on spare space in the Level 5 RAID is so fast (circa 20 sec for the RAID with ACATS after a disk failure and circa 101 sec otherwise) that another unrelated failure during this short period of time is not likely enough to impact overall reliability. The other fact is the better use of spare space in the Level 5 organization: If three or more failure happen in one reliability group, the Level 5 RAID can adapt to the situation whereas the MDS RAID becomes a victim of the fixed assignement of check disks to reliability groups. Reconfiguration of Reliability Groups (RRG) gives MDS based RAIDs the same flexibility. Consequentially, not only exhibit MDS RAIDs better resilience against multiple related failures, but they show essentially the same, though in fact a little better, resilience against unrelated failures.

We use a second level of reconfiguration, reconfiguration on check, to improve performance of MDS RAIDs. If a disk in an MDS RAID has failed, we reconstruct message data on space previously devoted to check data. Reconfiguration on check has no impact on reliability but makes reconfiguration of reliability groups feasible.

Reconfiguration for the extension of the classic scheme takes place when one reliability group has lost two check disks while there are other reliability groups which still have access to all their data. Then one check disk of a failure free reliability group serves as spare space for the affected reliability group. The result of the reconfiguration is two groups that can tolerate one more failure as opposed to one group which cannot tolerate further failure and one that can tolerate two failures.

Reconfiguration for the MDS RAID with ACATS mimicks the use of spare space in the Level 5 RAID with ACATS and a full distributed string of spares as well. If there is a disk failure, then all disks in the string containing the failed disk are on demand spare disks. In both cases, the reliability of the MDS RAID with reconstruction is the same as the Level 5 RAIDs with a distributed spare string and without or with ACATS.

4.6 Modeling in Detail

4.6.1 MDS Extension of the Classic RAID

The analysis of the scheme parallels closely the analysis in Section 3.4.1. The main difference is that there are now two different levels of vulnerability, namely loss of one disk and loss of two disks due either to string or to individual disk failure.

We characterize states by the number v of level 1 vulnerable reliability groups (one lost disk) and by the number w of level 2 vulnerable reliability groups and write (v, w) for these states. The number of failed disks in state (v, w) is $v + 2w$. In addition, we introduce states (S, v) to describe the states after a string failure, the other index indicates the number of *vulnerable* disks. Finally, state (T) describes the RAID after two string failures.

As usual, the state transitions corresponding to repair is taken with marginal probability ρ from every state to the normal state. The failure of essential equipment is modelled by a transition from each state to the failure state taken with probability ϵ.

A disk failure leads from state (v, w) to state $(v + 1, w)$ with probability $(m - v - w)n\lambda$, to state $(v - 1, w + 1)$ with probability $v(n - 1)\lambda$, and to the failure state with probability $w(n - 2)\lambda$, depending whether the failing disk is located in a not yet vulnerable reliability group, a level 1 vulnerable group or a level 2 vulnerable group. respectively.

The effect of string failure on a RAID in state (v, w) depend on the location of the

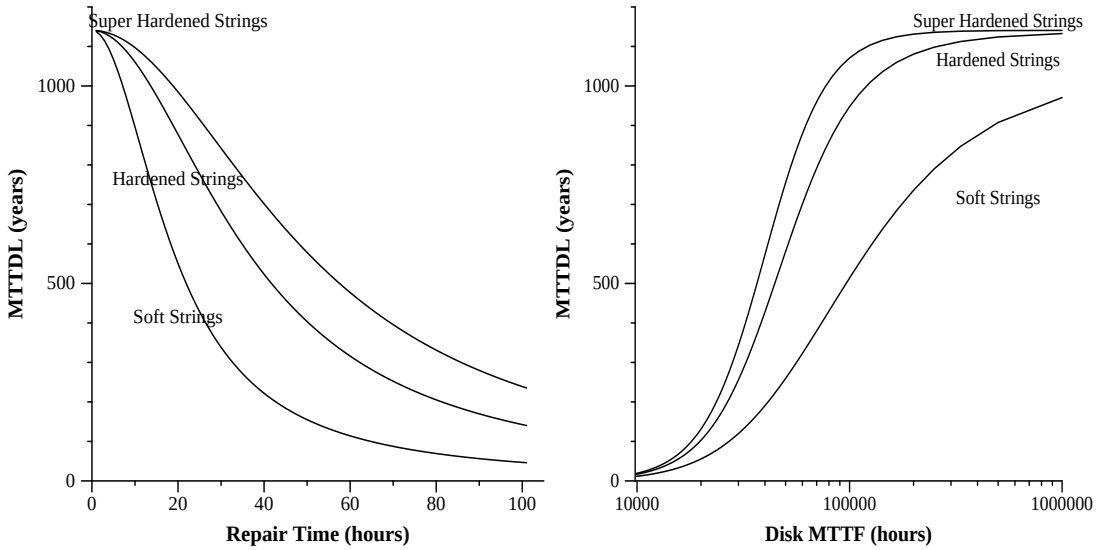


Figure 4.3: Impact of Repair Time and Disk MTTF on the MTTDL of the MDS extension of the Classic RAID

previously failed disks on the failing string. First we check the level 2 vulnerable reliability groups: There are two good choices for the failing string in each reliability group. With probability $(2/n)^w$ a string failure will not lead to data loss and with probability $1-(2/n)^w$ it will. Then we check the level 1 vulnerable groups: With probability

$$p(i,v) = \left(\begin{array}{c} v \\ i \end{array} \right) (1/n)^i ((n-1)/n)^{v-i}$$

i of the failed disks in the level 1 vulnerable groups will be located on the disks. Thus, we observe a transition from state (v,w) to state $(S, w+v-i)$ with probability

$$(\frac{2}{n^w}) \cdot p(i,v) \cdot n \cdot \mu$$

and with probability

$$(1 - \frac{2}{n^w}) \cdot n \cdot \mu$$

to the failure state.

To actually program the Markov chain, we use a state encoding function

$$(v,w) \rightarrow v + \frac{(2m - w + 3) * w)}{2}$$

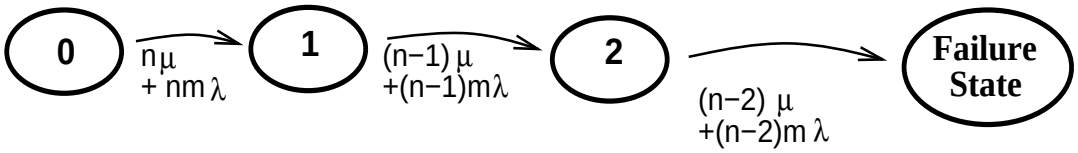


Figure 4.4: Markov Model for the MDS Extension of the Level 5 RAID with Almost Complete Address Translation

that maps the double indexed disk failure states into the coefficient space for the transition probability matrix M. Similarly, we map the single string failure states via

$$(S, v) \to v + \frac{(m+1)(m+2)}{2}$$

This allows us to automatize the calculation of the transition probabilities.

4.6.2 MDS Extension of the Level 5 RAID with Almost Complete Address Translation

This organization loses data if disks on three different strings are lost. We can capture the state of the RAID in only four different states, states 0,1 and 2, where the label is the number of strings with at least one lost disk. The simple Markov chain is depicted in Figure 4.4.

The transition matrix is

$$M = \begin{pmatrix} -mn\lambda & & \\ -n\mu - \epsilon & \rho & \rho \\ & & \\ mn\lambda + n\mu & -m(n-1)\lambda - (n-1)\mu & \\ +n\mu & -\epsilon - \rho & 0 \\ & & \\ & m(n-1)\lambda & -m(n-2)\lambda - (n-2)\mu \\ 0 & +(n-1)\mu & -\epsilon - \rho \end{pmatrix},$$

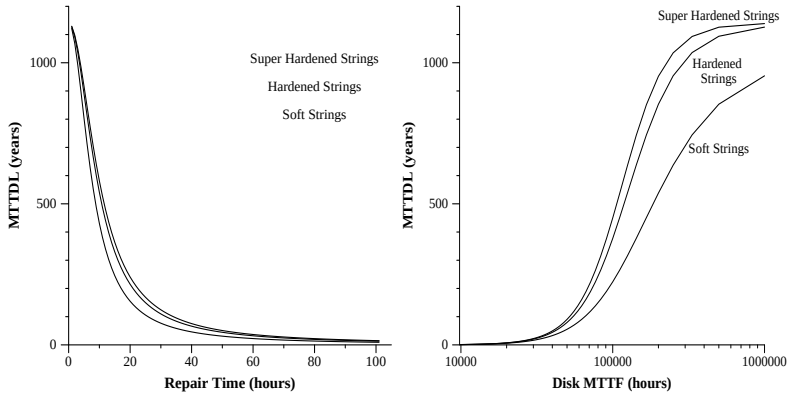


Figure 4.5: Impact of Repair Time and Disk MTTF on the MTTDL of the MDS RAID with ACAT

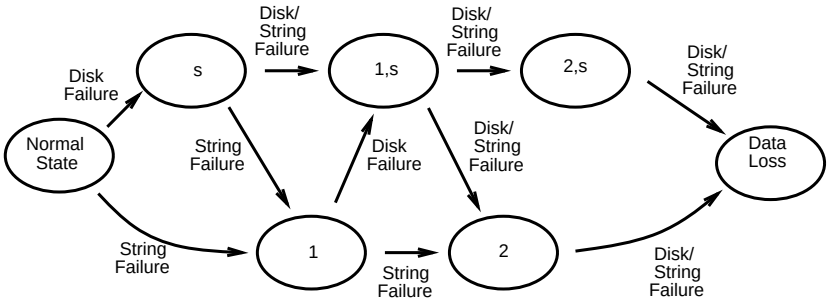


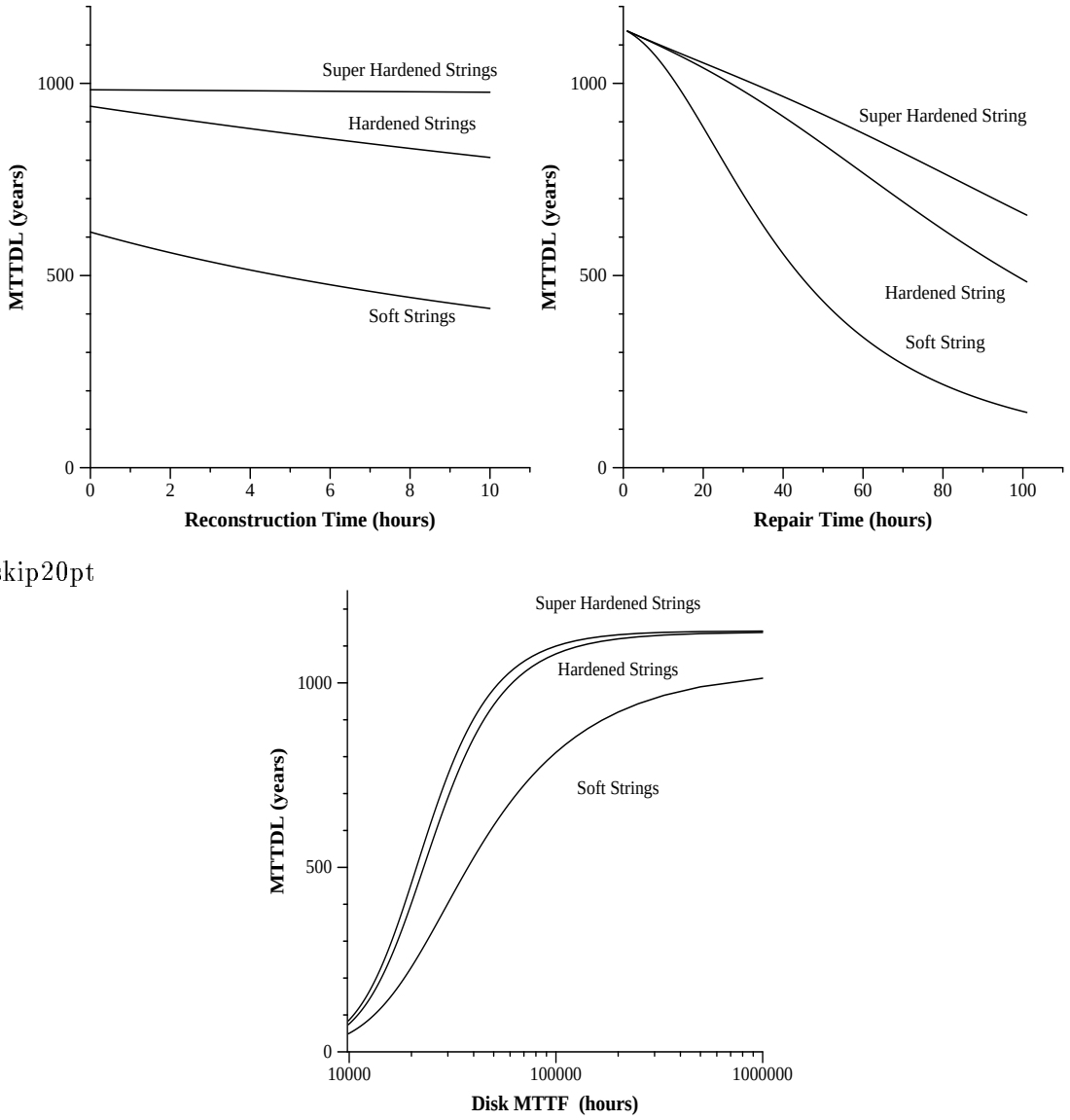
Figure 4.6: Markov Model for the MDS RAID with ACATS and SDS

from which we could easily calculate a closed form expression of the MTTDL value, which is not very readable. We give a sensitivity analysis in Figure 4.5.

4.6.3 MDS RAID with Almost Complete Address Translation and Safe Distributed Sparing

We provide one disk worth of spare space among all message disks. The states of the Markov model capture the number of strings that have failed or contain a failed disk. This number ranges from zero to two. In addition, we note, whether the RAID makes use of the spare space. We picture the Markov model in Figure 4.6. We distinguish the normal

and the failure state as well as state (s) in which a failed disk is reconstructed in the spare space, state (1), representing one string failure, state (1,s) depicting string failure and use of the spare space for an additional disk failure, state (2), two string failure, and state (2,s), two string failures and use of the spare space by an additional disk failure, which must have occurred before the last string failure. The transition rates are obtained from simply counting the remaining devices.



vskip20pt

Figure 4.7: Impact of Reconstruction Time, Repair Time and Disk MTTF on the MTTDL of the MDS RAID with ACAT

Chapter 5

Two Dimensional RAID Schemes

Gibson's thesis [11] advocated a two dimensional RAID scheme as a means to further increase RAID reliability. In this scheme each disk is part of two reliability groups, which overlap in exactly this one disk.

We show such an organization in Figure 5.1. There 16 information disks are arranged in a two-dimensional grid. Each vertical and horizontal axis forms together with the disk at the very right or bottom a reliability group of five disks each. The parity of the information disks is stored in the check disks (marked with a "C".) The poor ratio of 8 check disks versus 16 information disks improves as more disks are incorporated in the scheme. The shadings indicate disks belonging to a given string (consisting of four disks each.) The particular assignment of disks to strings - what we call the "Stringing the RAID" - is a difficult problem in Mathematical Combinatorics, if reliability is to be optimized. The naive stringing shown in the figure is not particularly good.

As there is only one check disk in each reliability group, we can use parity as the check. Furthermore, by using our *Almost Complete Address Translation Scheme* (ACATS) we can distribute the amount of check data, a physical disk carries, equally throughout the RAID and obtain good performance both in the absence of failed components and in the presence of a failed disk. The only alternative seems to be the use of a static assignment of disks, which severly limits performance because the checks will become hot spots during a write burst. The equivalent one dimensional scheme is known as a Level 4 RAID. We can use distributed sparing in both versions to bolster the RAID reliability.

The remainder of this chapter is organized as follows: In Section 5.1 we investigate

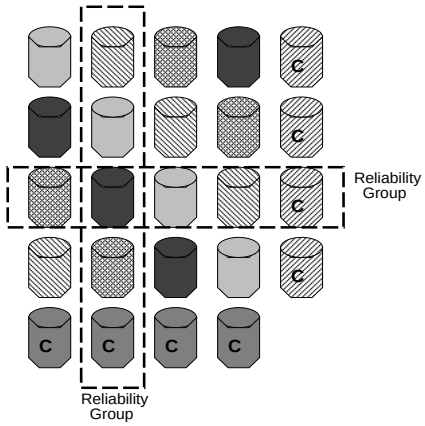


Figure 5.1: A Two Dimensional RAID Scheme

the resilience of the twodimensional scheme against disk failure only. Our results indicate the good potential of the scheme. The model relies on calculating numbers of *Disk Failure Patterns.* This task has been relegated to Section 5.3, which is quite technical and can be skipped. In Section 5.2 we discuss the problem of assigning disks to strings. The best stringing scheme will offer tolerance against simultaneous failure of two strings (or disks.) We can show that without sparing no scheme can exist that exceeds this tolerance. We conjecture that stringing schemes that offer tolerance against two component failure and attain the obvious bound of n disks per string (for a total of $n^2 + 2n$ disks) do not exist for $n \geq 5$. We give, however, schemes with a smaller number of disks per string that exhibit tolerance against two component failure (Section 5.5.2) and we calculate their reliability in Section 5.5.3. The twodimensional RAID uses a somewhat larger number of disks than most MDS code based RAIDs with same capacity and the last mentioned schemes use more strings. The reliability numbers in Table 5.5 can only be compared with this *caveat* in mind.

5.1 Disk Failure Reliability of the 2 Dimensional RAID

We first investigate the degree to which a 2 Dimensional RAID can achieve data safety against disk failures. We investigate a RAID that uses a full address translation scheme, in which a logical disk track address is mapped potentially into every physical disk track address. In Table 5 we give the data loss probabilities for such a RAID with a given

Number	48 Disks Static	48 Disks Balanced	120 Disks Static	120 Disks Balanced
3	0.0020814061	0.87551565	0.0003560746	0.29962407
4	0.0104070305	0.99995434	0.0017255925	0.82219770
5	0.0315364561	$1 - 1.21 * 10^{-14}$	0.005374177	0.995431954
6	0.0745336030	$1 - 2.29 * 10^{-34}$	0.012580325	0.999996825

Table 5.1: Data Loss Probabilities for the Static (no address translation) and the Balanced RAID (full address tranlation) for two RAID Sizes with Capacity 36 Disks and 100 Disks.)

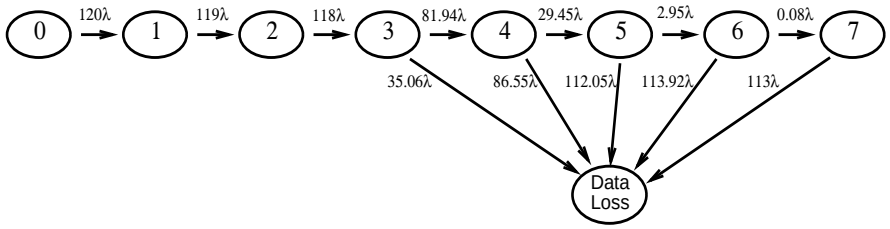


Figure 5.2: Markov Model for the Disk Failure Reliability of the Two Dimensional RAID with 120 Disks.

number of disk failures. As we can see, the disk address translation scheme has a strong negative impact on the reliability. Failure of 5 disks for the small (48 disks) and 6 disks for the larger (120 disks) RAID lead to data loss with a probability that is not distinguishable from one.

To derive the results, we first calculate the data loss probabilities for a "static" RAID without full address translation. This RAID organization is hardly usable, as the performance is severly limited by the load of the check disks. In contrast to the static RAID, a RAID with address translation consists of 1000 (the number of tracks) logical static RAIDs and the calculation of the data loss probabilities reflects the fact that data loss has to be avoided for all these 1000 virtual RAIDS.

In Section 5.3 we give the details of the enumeration of all "disk failure patterns" that lead to data loss in the static RAID. To summarize the method there, we count the *Minimal Disk Failure Patterns,* which are not contained in any disk failure pattern, that leads to data loss. Failure of three disks is necessary to induce data loss, but only if they

Disk Failure Probability (per hour)	Two-Dim. RAID 48 Disks	Two-Dim. RAID 120 Disks	MDS RAID 48 Disks	MDS RAID 120 Disks
0.000005	1171.64	1153.92	1051.47	387.48
0.000010	1152.22	937.08	616.04	71.93
0.000020	922.99	252.53	148.13	10.62
0.000040	242.88	24.06	22.67	1.65
0.000080	23.76	2.26	3.40	0.30

Table 5.2: Reliability against Disk Failure of the Two-Dimensional RAID (with 36 and 100 disks capability) compared with 2 MDS RAIDs (with Dimensions 6 * 8 and 10*12). The survival rate is given in years.

form a certain pattern, the "open triangle" (see Figure 5.3). If we consider failure patterns for disks that consist of more than three disks, then a majority of those will consist of an "open triangle" and other disks. Similarly, any failure pattern will contain at least on minimal failure pattern. We can use the combinatorial principle of inclusion and exclusion together with the count of minimal failure patterns to count all failure patterns with a given number of disks. This gives us the probability of data loss for any given number of disk failures. Because this probability becomes overwhelming for 7 disk failures or more, we only enumerate failure pattern containing up to 6 disks.

The resulting Markov model, which takes only disk failures into account, (see Figure 5.2) is easy to derive. We only need to consider repair, disk failure and essential component failure transitions. The non-absorbing states of the model reflect the number of failed disk. We calculate the transition rates are from the data loss probabilities given the failure of that number of disks.

We give the survival rates for a range of disk failure probabilities in Table 5.1; the third value is our standard value. The high reliability of the two-dimensional scheme is apparent. We should note, that a non-hashed MDS-RAID shows much higher reliability. We have results about the tremendous increases in reliability available with the use of distributed sparing in Two-Dimensional RAIDs. While these numbers do not apply to any existing RAIDs, they show the reliability promises of the two dimensional RAIDs.

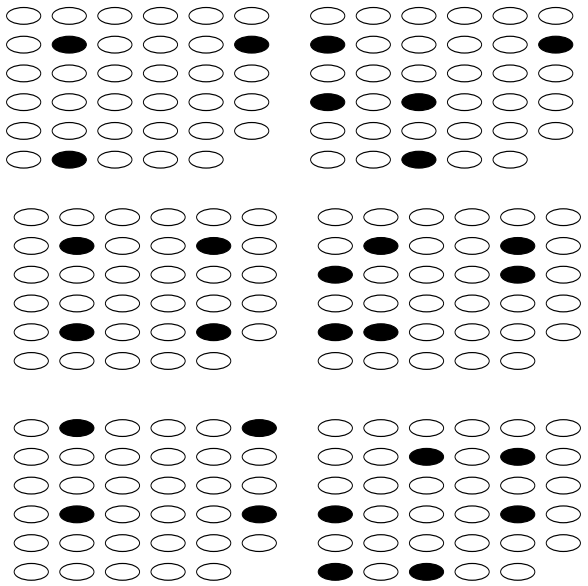


Figure 5.3: Minimal Disk Failure Patterns: Open Triangle, Closed Quadrangle, Open Quadrangle, Open Pentagon, Closed Hexagon, Open Hexagon

5.2 Stringing the Two-Dimensional RAID

5.2.1 Figures of Merits for Stringing Schemes

In contrast to the One-Dimensional RAID schemes and to MDS-RAIDs, there is no natural way to group individual disks into strings. The resulting freedom of design beckons the question about the definition of goodness of a stringing scheme. Reliability as expressed in data life expectancy suggests itself, but is not the only figure of merit: we have also to include performance, both in the normal case and under various probable failure conditions, and, a related issue, the storage demands of data reconstruction. Some of the stringing schemes, that we will consider, illustrate the point, for it might be necessary to repeatedly invoke the data reconstruction process in order to access data lost due to an almost catastrophic series of failures and, in the process, severly strain the storage capacities of the RAID as well as restrict performance of access to data not directly involved in the failures.

Development of a stringing scheme is a combinatorial problem of great difficulty. We can derive more accessible figures of merit by measuring reliability in how many string

and disk failure a RAID with a particular stringing scheme can survive and how many reconstructions step might be necessary to recover data in the worst assumed case. We demand that strings have the same size, to allow for load distribution schemes based on address translation. While a final design might use more than our three level hierarchy of support (full RAID, strings, disks), we want to maximize the size of the string to make best use of reliable power and cooling.

5.2.2 Maximum Tolerance against String Failures

The minimum number of disks, whose failure induces data loss is three. In section 5.3 we introduced the notion of a triangle, a set of three disks, one of which is an information disk and the other two of which are check disks located in the same row and in the same column. (See Figure 5.3, upper left corner.) Inavailability of the disks in a triangle constitutes data loss. If all three disks in a triangle are located on different strings, then the failure of the three strings implies data loss. If all or two of them are located in the same string, then failure of one or two strings can lead to data loss. Hence we can draw the easy conclusion that in any stringing scheme failure of any three components causes data loss.

5.2.3 Tolerance against Single Disk and String Failure

Tolerance against single disk and simultaneous string failure is a minimal requirement for two dimensional RAIDs. We derive a mathematical criterion for such a stringing scheme in Section 5.4. Schemes exists for any parameter n. We give an example in 5.1.

5.2.4 Tolerance against Two String Failures

The stringing scheme with the highest reliability exhibit tolerance against simultaneous failure of two strings and, consequentially, against failure of any two components. We present stringing schemes with that level of tolerance in Section 5.5. While we are only capable of presenting schemes with the maximum number n disks per string for unusable small RAIDs, we can find manually examples with smaller string size for most important values of n. These schemes actually present the advantage of less reconstruction steps in the case of two string failure. We investigate the reliability of RAIDs given there in Section 5.5.3.

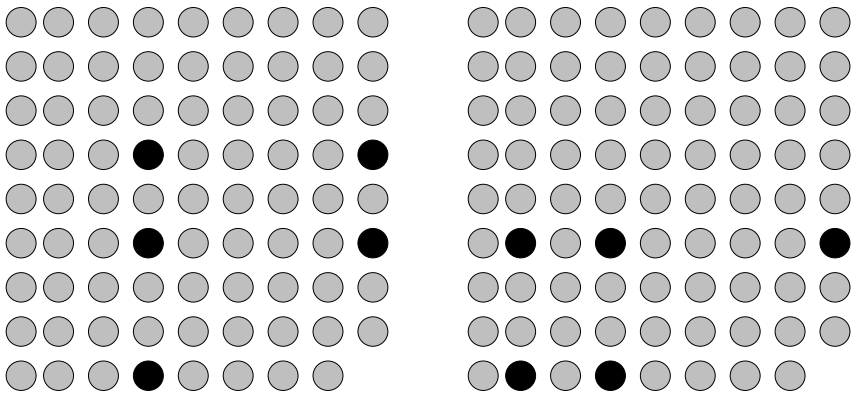


Figure 5.4: Two 5 DFPs Containing Two Triangles and a Quadrangle.

5.3 Disk Failure Patterns

5.3.1 Definition

A *Disk Failure Pattern* (DFP) is a set of disks in the RAID, whose failure would lead to irrecoverable data loss. A *Minimal Disk Failure Pattern* (MDFP) is a disk failure pattern that does not contain a smaller disk failure pattern. For example, the set of all disks is trivially a disk failure pattern, but not a minimal disk failure pattern.

It is easy to describe all MDFP. A n-gon (with $n \geq 3$) is a set of disks in the RAID, that are the vertices of a path through the RAID, which alternatively follows the rows and the columns. In a *closed n-gon,* all the disks in the n-gon are information disks and the path is circular. In an *open n-gon* the endpoints of the path are check disks and the middle points are information disks. We give examples for the smallest n-gons in Figure 5.3. A fundamental result is now that the MDFPs are exactly the open and closed polygons. The proof is straightforward, but not illuminating, and hence omitted.

5.3.2 Enumeration of MDFP and Reliability Bounds

We count the number of MDFP with a given number of elements for a two-dimensional RAID with $n \times n$ information disks (and a total of $n^2 + 2n$ disks.) Given this number and the total number of failure patterns for disks, we calculate the reliability of a static RAID and a RAID with complete address translation. We give the data loss

probabilities for RAIDs with 36 and with 100 information disks (and 48 and 120 resp. disks total) for comparison purposes. At this point, we neglect string failure completely, hence we are giving optimistic bounds. The numbers for the balanced RAID (full address translation) are based on 1000 tracks.

3 Disks Failure Patterns: The only disk failure patterns with three disks are the open triangles. There are n^2 of these, corresponding to the choice of the one information disk, whereas there are $\begin{pmatrix} n^2+2n \\ 3 \end{pmatrix}$ possible patterns of three failed disks.

4 Disks Failure Patterns: A 4 DFP either contains an open triangle, when it is not minimal, or is an open or closed polygon. There are n^2 open triangles and an additional choice of a single disk among all those not on the triangle yields a count of $n^2(n^2+2n-3)$ for the 4 DFP containing an open triangle. There are two kinds of open quadrangles; one kind has the end points in the right check disks, the other in the lower check disks. A quadrangle of the first kind is determined by one column and then by two disks in this column. We count $n^2(n-1)/2$ for the first kind and $n^2(n-1)$ for all of them. A closed quadrangle is determined by two columns and two rows of information disks. Hence, there are $n^2(n-1)^2/4$ of them. The total number of 4 DFP is

$$\#(4\text{ DFP}) = n^2(n^2+2n-3) + n^2(n-1) + n^2(n-1)^2/4$$

5 Disks Failure Patterns: There is only one 5 MDFP, the open pentagon. There are characterized by first choosing the middle vertex and then another information row and column. We count $n^2(n-1)^2$. Non minimal 5 DFPs contain either a 3 or a 4 MDFP. We counted $n^2(n-1)+n^2(n-1)^2/4$ quadrangles. We multiply this number with the possibilities for picking an additional disk and obtain $n^2(n-1)(1+\frac{n-1}{4})(n^2+2n-4)$ as the count for 5 DFPs containing a quadrangle. Alternatively, a 5 DFP contains a 3 MDFP. If we just enumerate $n^2(n^2+2n-3)(n^2+2n-4)/3$ we count certain failure patterns several times. This type is exemplified in Figure 5.4. (The circles represent disks, the last row and the last column are the check disks.) This class contains failure patterns with two triangles and a quadrangle. They are given by either one of n rows and 2 of n columns or vice versa. This

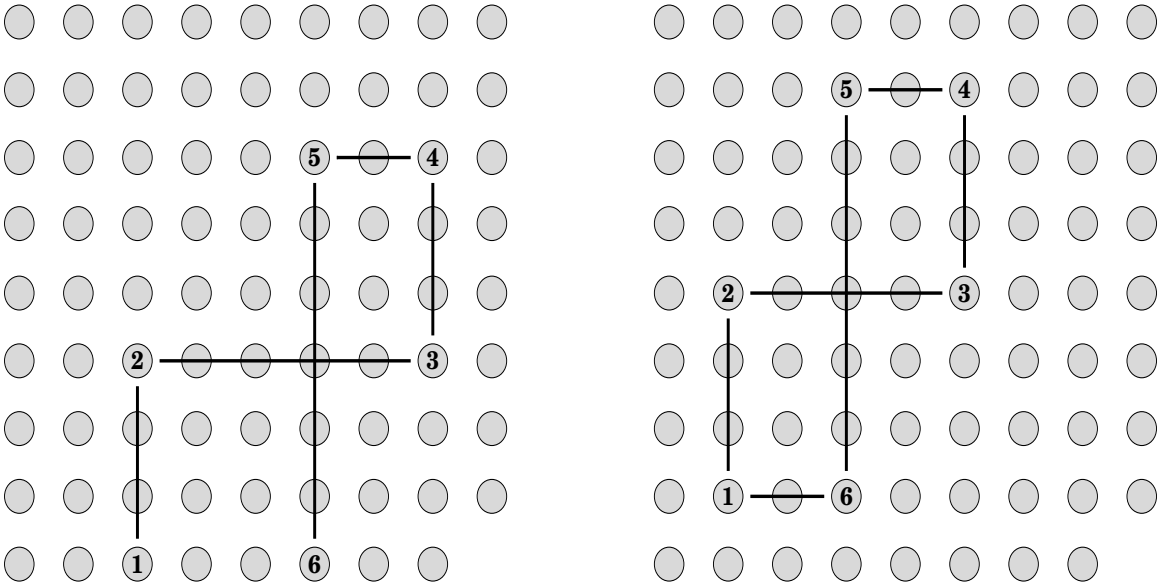


Figure 5.5: Calculation of 6 MDFP

puts their number at $n^2(n-1)$. Our grand total for 5 DFPs is

$$\begin{aligned}\#(5\text{ DFP}) &= n^2(n-1)^2 + n^2(n-1)(1+\frac{n-1}{4})(n^2+2n-4) \\ &\quad + n^2\frac{(n^2+2n-3)(n^2+2n-4)}{2} - 2n^2(n-1)\end{aligned}$$

6 Disks Failure Patterns: An open hexagon whose end vertices are in the row of check disks is determined by (1) choosing a starting disk in this row of check disks, (2) choosing another disk in the same column, (3) choosing another disk in the same row, (4) choosing another disk in the same column, which has to be in a different row than the one picked in step (2), (5) choosing another disk in the same row, which again has to be in a different column than chosen in step (3). There are hence $2n^2 * (n-1)^2 * (n-2)$ possible paths through the RAID defined by our procedure. As a hexagon can be traversed two ways, counting the possible paths, as we just did, overcounts by a factor of two. On the other hand, there are as many open hexagons based on the lower check disks as there are those based on the right check disks, and we obtain a total of $2n^2 * (n-1)^2 * (n-2)$ for the open hexagons. (The counting procedure is illustrated in Figure 5.5.)

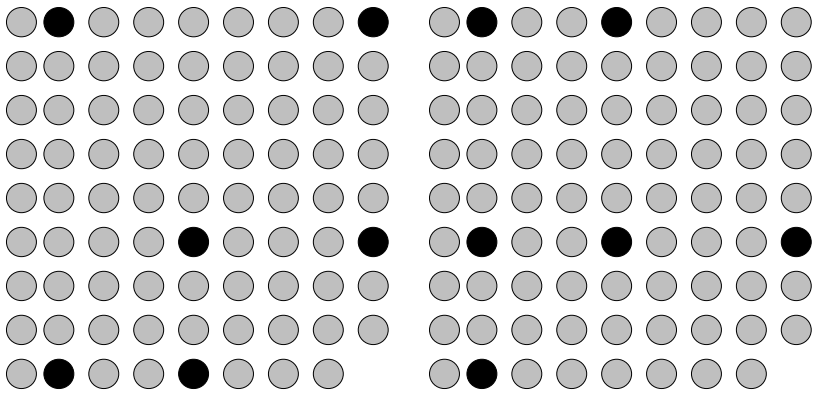


Figure 5.6: 6 DFP with Two Disjoint Triangles and with One Quadrangle and One Triangle

A closed hexagons is similarly given by a path. (1) We pick one information disk, then (2) another one in the same column, from there, (3) a third one in the same row, (4) the fourth one in the same column, but not in the same row as the first and the second disks, and (5) the fifth one in the same row, but not in the same column as the second and third disks. The sixth disk then has to be located in the same row as the first one and in the same column as the fifth disk. The number of paths defined in this manner is $n^2(n-1)^2(n-2)^2$. For any given closed hexagon, there are six different paths (which only differ in orientation and starting point) enumerating the vertices of the hexagon. We have to divide our paths count by six to obtain the number of closed hexagons.

We now need to count the number of 6 DFPs which are not minimal. Without adjustment for overcounting, we obtain

$$\begin{aligned} & n^2\frac{(n^2+2n-3)(n^2+2n-4)(n^2+2n-5)}{6} \\ + \quad & n^2(n-1)(1+\frac{n-1}{4})\frac{n^2+2n-4)(n^2+2n-5)}{2} \\ + \quad & n^2(n-1)^2(n^2+2n-5) \end{aligned}$$

non-minimal 6 DFPs. The summands in the formula corresponds to triangles, quadrangles and pentagons. Adjusting for the overcounting is an arduous procedure.

First, there are 6 DFPs with two triangles. As we have seen previously, there is a class of 5 DFPs that contain two triangles and a quadrangle. By adding one more failed

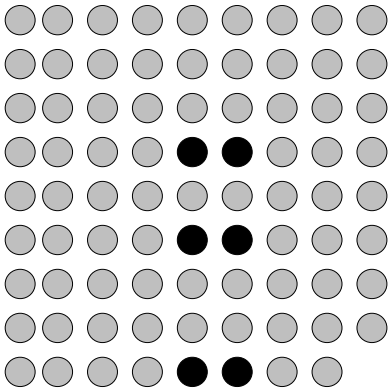


Figure 5.7: 6 DFP consisting of two quadrangles

disk, we count $n^2(n-1)(n^2+2n-5)$. We have to diminish our total by twice this number, because we counted them three times in our naive formula. The other possibility for a 6 DFP with two triangles consists of two disjoint triangles. (See Figure 5.6, left.) There are $n^2(n-1)^2/2$. Another set of overcounted 6 DFPs consists of a closed quadrangle and a triangle. (See Figure 5.6, right.) At the same time, they also contain a pentagon. There is one possibility to install a triangle on top of a quadrangle for each of the four quadrangle corners, so that we put the number of these patterns at $n^2(n-1)^2$. We have to subtract twice this amount from our grand total. We give an example for a 6 DFP consisting of two quadrangles in Figure 5.7. They are enumerated by either choosing three of $n+1$ rows and two of n columns or two of n rows and three of $n+1$ columns. The count is $(n+1)n^2(n-1)^2$. After taking all these adjustments into account we enumerate

$$
\begin{aligned}
& \#(6 \text{ DFP}) = \\
& n^2(n-1)^2(n-2) + \frac{n^2(n-1)^2(n-2)^2}{6} \\
+ \quad & n^2 \frac{(n^2+2n-3)(n^2+2n-4)(n^2+2n-5)}{6} \\
+ \quad & n^2(n-1)(1+\frac{n-1}{4})\frac{n^2+2n-4)(n^2+2n-5)}{2} \\
+ \quad & n^2(n-1)^2(n^2+2n-5) \\
- \quad & 2n^2(n-1)(n^2+2n-5) - \frac{n^2(n-1)^2}{2} \\
- \quad & 2n^2(n-1)^2 - (n+1)n^2(n-1)^2
\end{aligned}
$$

for the number of 6 DFPs.

5.4 A Stringing Criterion for Tolerance against One Disk and One String Failure

We first introduce cartesian coordinates to describe the location of a disk with respect to reliability groups. Let disk $D_{i,j}$ denote the disk in the i^{th} horizontal reliability and j^{th} vertical reliability group. We designate with $D_{i,n+1}$ the check disk in the i^{th} horizontal reliability and with $D_{n+1,j}$ the one in the j^{th} vertical reliability group. We first need a technical definition:

Definition 1 *A* **Triad** *is a set of three disks* $\{D_{i,j}, D_{j.k}, D_{l,j}\}$ *with (pairwise) different* i, j, k, l.

Every triangle is a triad, but a triad is a triangle only if the two "end-pieces" are check disks. We can now formulate our criterion:

Theorem 6 *A stringing which allows the RAID to tolerate simultaneous failure of any single disk and any single string failure is characterized exactly by the following properties:*

1. *No string contains a triad.*
2. *No string contains both a check disk* $D_{i,n+1}$ *and a check disk* $D_{j,n+1}$.
3. *If a string contains a check disk* $D_{i,n+1}$ *it does not contain any disk* $D_{i,j}$ *in the same row.*
4. *If a string contains a check disk* $D_{n+1,j}$ *it does not contain any disk* $D_{i,j}$ *in the same column.*

Proof: The proof makes use of the simple observation, that any failure pattern that leads to data loss contains an information disk, whose contents cannot be restored, and that this implies that another disk in the same row or column has failed.
"$\Rightarrow$": Assume that a stringing shows the necessary failure tolerance. We show, that everyone of the criteria cannot be violated.

1. If a string contains a triad, assume that the string has failed. If the triad is already a triangle, the RAID experienced data loss. If not, then complete the triad to form a quadrangle and assume that the additional disk has failed. This leads to data loss.

2. If a string contains both a check disk $D_{i,n+1}$ and $D_{n+1,j}$ then failure of the string and disk $D_{i,j}$ causes data loss.

3. If a string contains the check disk $D_{i,n+1}$ and the information disk $D_{i,j}$ then failure of this string and $D_{n+1,j}$ gives raise to data loss.

4. The same argument applies *mutatis mutandis* to columns.

"$\Leftarrow$": Assume that data loss in disk $D_{i,j}$ has been occasioned by the simultaneous failure of a string and a disk. We now distinguish several cases, which will show that one of our criteria has been violated.

Case 1: Assume that $D_{i,j}$ was the additional disk. The string then contained disks in the same row and the same column.

Case 1a: One of these disks is a check disk.

Let us assume that $D_{i,n+1}$ is in the string. (The column case is treated strictly analogous.) Another disk $D_{l,j}$ has also to be on the string, or data $D_{i,j}$ would not have been lost. If this disk is a check disk ($l = n + 1$) criterion 2 is violated. If not, then $D_{l,j}$ is an information disk and another disk $D_{l,k}$ has to be in the string. But then the three disks $D_{i,j}$, $D_{l,j}$ and $D_{l,k}$ form a triad.

Case 1b: Disk $D_{i,j}$ has failed and information disks $D_{i,l}$ and $D_{k,j}$ are located on the string.

We conclude that in both the l^{th} column and the k^{th} column another disk of the string is located. Unless criterion 2 is violated, at least one of these is not a check disk. We treat only the case that the information disk $D_{i,m}$ is situated on the string. Because the data on this disk cannot be restored, A further disk in the same vertical reliability group has to belong to the string, so that the string contains a triad.

Case 2: $D_{i,j}$ is based on the string.

Then there are disks $D_{i,k}$ and $D_{l,j}$ which have lost data as well. If both belong to the string, the string contains a triad. So assume, that $D_{i,k}$ is the additional failed disk. (The other case is treated analogously.) We find ourselves back in case 1 unless $D_{i,k}$ is a check disk, that is $k = n + 1$. Either $D_{l,j}$ is a check disk as well (in violation of criterion 4) or it is an

information disk, whose data cannot be restored. In the latter case, another disk in row l has lost data and must be on the string, which then contains a triad. We have accounted for all possible cases. ■

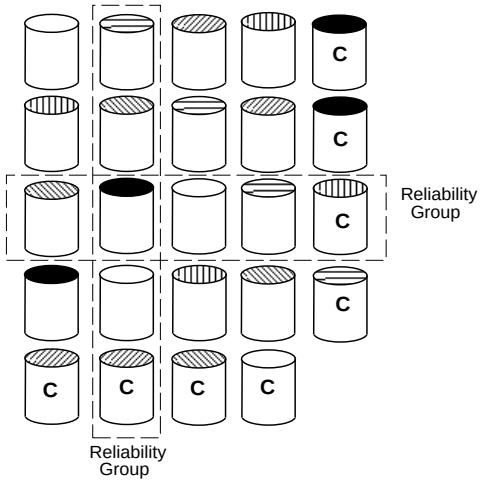


Figure 5.8: Walt Burkhard's Stringing Scheme for a Two-Dimensional RAID Tolerating Failure of Two Strings.

We can use the theorem to give an upper bound on the number of disks in any stringing arrangement, even those without equal number of disks in a string. Because no string can contain a triad, once a disk in a certain row and column has been placed into the string, then either the row or the column can contain no further disk. The shape of the largest string thus consists of a row with the exception of one disk and the column where this disk is lacking. Thus the string size cannot exceed $2n$. As the RAID cannot be tiled with strings of the proposed form, we cannot exceed a fixed string size of $2n - 1$. The real number is much smaller, as it needs to be a divisor of $2(n + 1)$.

5.5 Criterion for Tolerance against Two String Failures

A stringing scheme with the stated tolerance against two string failures satisfies *a fortiori* the criteria laid out in Theorem 6. As the number of disks lost in a failure that did not lead to data loss cannot exceed $2n$, the level of redundancy, a stringing scheme with equal numbers of disks per string cannot exceed n.

1	2	3	4	5	6	9
7	1	2	3	4	5	9
8	7	1	2	3	4	10
3	8	7	1	2	6	10
10	6	5	7	8	12	4
6	12	9	5	9	8	10
11	11	12	12	11	11	

Figure 5.9: A stringing scheme for the two-dimensional RAID scheme with 36 disks capacity, that tolerates failure of two strings and consists of 12 strings with 4 disks each.

5.5.1 Schemes with Maximum Number of Disks in a String

We are only able to give organizations for small numbers of n. The biggest organization was found by Walt Burkhard and is presented in Figure 5.8. We conjecture, that this is the maximum number n for which such a scheme exists. We have written a search program that starting from an assignment with a high degree of tolerance against two string failures tries out random alterations and that so far has performed unsuccessfully.

5.5.2 Schemes with a Smaller Number of Disks in a String

Stringing schemes with a smaller number of disks per string constitute an alternative to ones with n disks per string. Because the number of disks in a string is a divisor of $n(n+2)$ solutions will depend on the prime factorization of n. If n is the lower of a pair of primes (e.g. 11) then this approach is impossible.

We have written a small computer tool, that checks the failure tolerance of a stringing scheme. Starting from a heuristic stringing assignment we then try to heal conflicts (that is: cases in which loss of two strings leads to data loss) by small permutations. For a design of a production line scheme, this semi-manual approach can clearly be transferred into a CAD tool based on Simulated Annealing and find an assignment maximizing secondary goals like the maximum or average number of data recovery operations. In general, we can observe that with this less ambitious approach, these two numbers are lower.

5.5.3 Markov Models for the Reliability of the Stringing Schemes offering Two String Failure Resistance

The Markov model for the reliability of a stringed RAID differs from the preliminary investigation of an unstringed RAID (see Section 5.1.) In a stringed RAID, failure of any number of disks located on only two strings can never lead to data loss, whereas the other model assumed that to happen with quite a large probability. If we know, however, that disks on more than two strings have failed, then we can give rather accurate data loss probabilities. Because the different schemes exhibit non-scalable parameters as the number of strings or the number of disks per string, we will give our calculations in Tables 5.5, 5.5.2 and 5.5.2.

First we derive the Markov model for the RAID with stringing scheme shown in Figure 5.5. As we had calculated before, there are 36 triangles. If three disks on three different strings have failed, (14080 cases), the probability that these three disks form a triangle is about 0.2% individually and about 92.3% for a RAID with address translation and 1000 tracks. The probability calculation is based on the assumption, that our address translation scheme maps every set of three disks on three different strings to another set of three disks on three different strings with the same probability. The analogue assumption for larger numbers of disks located in more than three disks remains only approximately valid. We give the results of our probability calculations in Table 5.3. (The last line in part (c) shows effects of rounding errors, that however do not effect the veracity of our Markov model based calculations.) The difference to the values in Table 5 are minimal. We then translate these probabilities into the transition probabilities to the Failure State from states in our Markov model, that only reflect disk failures.

In contrast to the "disk failures only" approach in Section 5.1 we need to include states in our Markov model that correspond to one string failure in conjunction with disk failures. As long as the additional disk failures befall only one additional string, data loss is impossible.

Let us first consider the easiest case of a string failure and two additional disk failures on two different strings. The failures make up a triangle, if either both additional string failure are check disks or if one is a check disk and the other one is located in the same row/column, and, in addition, if one of the disks on the failed string makes up the missing

(a)

Number of Disk Failures	Probability (static)	Probability with Address Transl.
3	0.002556818	0.922702404
4	0.010659466	0.999977828
5	0.031604675	1.0
6	0.074544829	1.0

(b)

Number of Disk Failures	Probability (static	Probability with Address Transl.
3	0.00094451	0.611303469
4	0.00395790	0.981046289
5	0.01179252	0.999992951
6	0.02783050	1.000000000

(c)

Number of Disk Failures	Probability (static	Probability with Address Transl.
3	0.00041834	0.341919198
4	0.00180099	0.835133097
5	0.00537842	0.995451416
6	0.01267302	0.999997109

Table 5.3: Data loss Probabilities of Disk Failures for the RAID with (a) 12 Strings with 4 Disks each, (b) 16 Strings with 5 Disks each and (c) with 20 Strings with 6 Disks each

vertex. If the stringing scheme consists of s strings of length l disks, then this probability is given by:

$$\text{Prob}_{\text{DI}} = \frac{3n \cdot n}{(n^2+2n)(n^2+2n-1)} \cdot \frac{1}{s-2}$$

where the first factor is the chance that either the two additional disks are both check disks or one is a check disk and the other is in the same reliability group. The probability applies to static RAIDs (without address translation) and needs to be translated into the much higher one We give the relevant probability for the RAIDs under consideration (Figures 5.5, 5.5.2 and 5.5.2.) While the effective data loss probability is high, we need to look at one case further, in order to maintain our high standards of accuracy for Markov models.

So, assume that three additional disks on at least two strings have failed. With probability

$$n^2 \cdot \left(\begin{array}{c} n^2+2n \\ 3 \end{array} \right)^{-1}$$

RAID Size	**String + 2 Disks**	**String + 3 Disks**
36+12 Disks	0.991759791	0.999999937
64+16 Disks	0.886089217	0.999598449
100+20 Disks	0.688954212	0.979035703

Table 5.4: Data Loss Probabilities for the three RAIDs with a String Failure and additional Disk Failures. (The third column is a Lower Bound.)

Disk Array Organization	MTTDL (in years) Super Strings	MTTDL (in years) Hard Strings	MTTDL (in years) Soft Strings
48 disks	869.91	374.02	120.86
48 disks (SDS)	1120.32	685.01	254.77
80 disks	448.47	129.26	36.91
80 disks (SDS)	1025.84	320.10	84.96
120 disks	213.95	52.35	14.70
120 disks (SDS)	803.62	137.82	33.37

Table 5.5: Reliability of the Three Two-Dimensional RAID schemes developed in this Section. Every other row gives the reliability for the RAID with SDS for one Disk.

these three disk failure will cause data loss by itself. We can estimate the probability of data loss caused with involvment of the failed string to be at three times (3 choose 2) the probability given in the preceding paragraph. This is a lower estimate that disregards quadrangles at all, but it is (barely) adequate to assume data loss in these cases for the RAIDs which use address translation. We thus can forego a more detailed but very difficult calculation.

The resulting Markov model is given for the largest of our three RAIDs in Figure 5.12. We only show transitions that have a non-negligible probability, under which we understand any marginal transition probability, that needs to places after the decimal point to be represented. We give the results of our reliability calculations for the three sample RAIDs in Table 5.5.

1	12	3	4	5	6	7	16	11
10	13	2	3	4	5	6	7	11
6	9	1	2	3	4	5	8	13
7	8	9	1	2	3	16	5	14
9	7	8	9	11	2	3	6	12
16	10	7	8	13	9	11	12	14
4	2	14	10	15	8	16	12	13
12	14	11	15	1	14	4	6	13
15	1	10	5	10	16	15	15	

Figure 5.10: A stringing scheme for the two-dimensional RAID scheme with 64 disks capacity, that tolerates failure of two strings and consists of 16 strings with 5 disks each.

14	15	3	4	5	6	18	9	17	10	11
12	1	2	3	4	5	6	7	8	9	17
18	16	1	2	3	4	5	6	7	8	9
12	11	14	1	2	3	4	5	6	7	13
9	10	11	13	1	2	3	4	5	14	7
8	17	10	16	12	1	2	3	4	13	14
19	18	9	13	11	16	7	8	14	15	10
11	19	12	14	15	18	16	17	1	12	8
6	20	19	16	17	15	10	13	18	6	11
10	12	17	18	16	7	8	19	13	15	9
5	2	20	15	19	20	20	20	20	19	

Figure 5.11: A stringing scheme for the two-dimensional RAID scheme with 100 disks capacity, that tolerates failure of two strings and consists of 20 strings with 6 disks each.

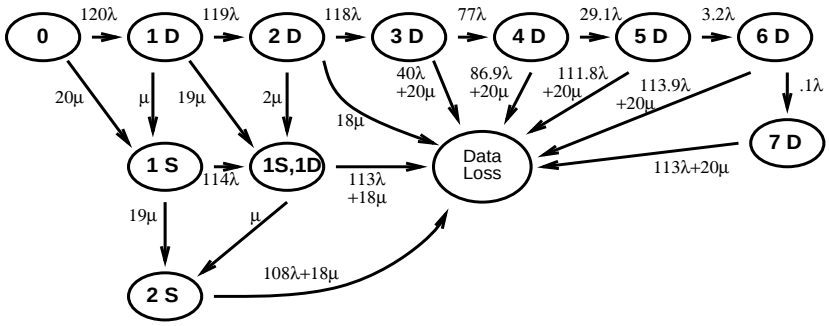


Figure 5.12: The Markov Model for the two-dimensional RAID with 20 strings with 6 Disks each. We only show transitions with non-negligible transitions.

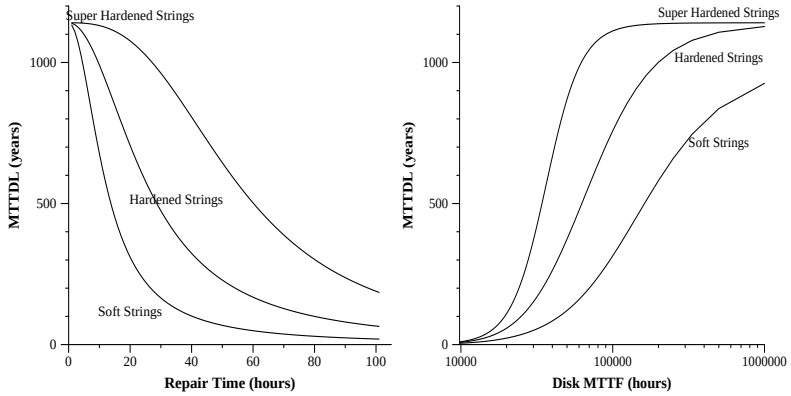


Figure 5.13: Impact of Repair Time and Disk MTTF on the Two Dimensional RAID with 48 Disks and 12 Strings

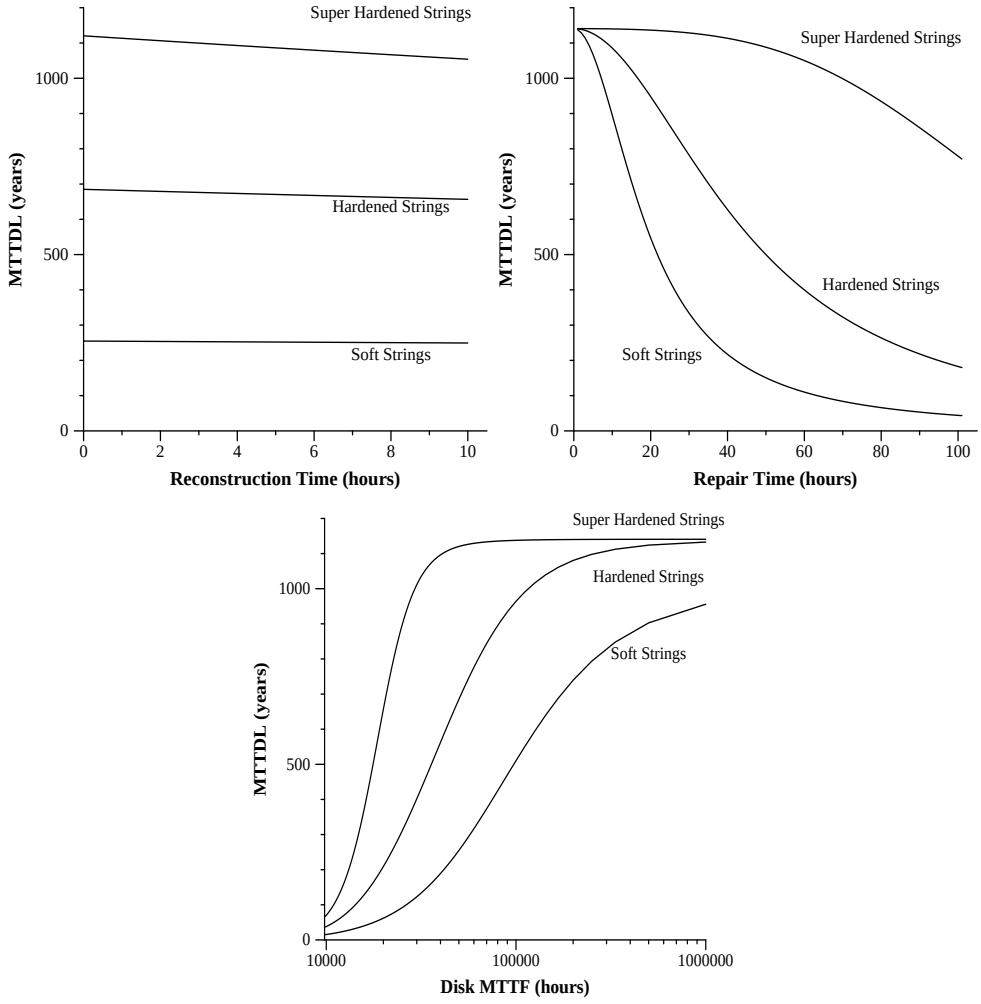


Figure 5.14: Impact of Reconstruction Time, Repair Time and Disk MTTF on the Two Dimensional RAID with 48 Disks and 12 Strings with SDS (1 Disk)

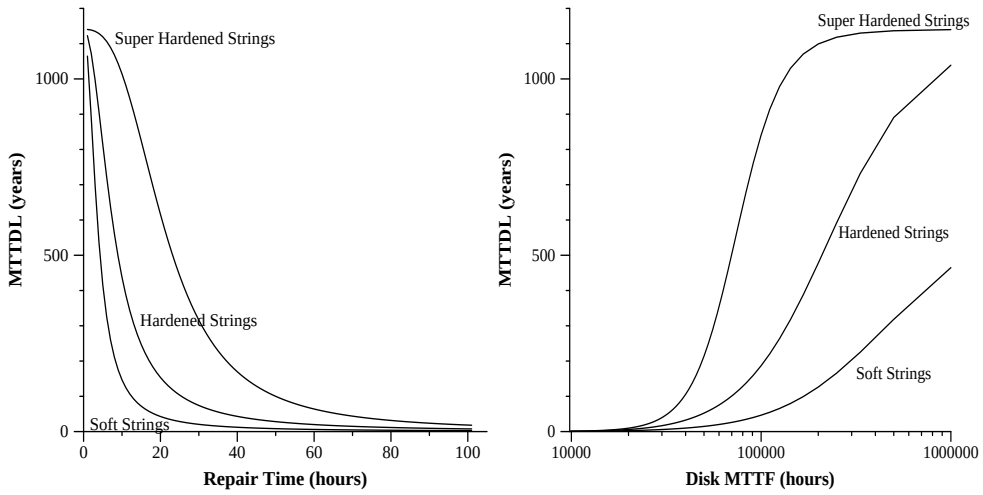


Figure 5.15: Impact of Repair Time and Disk MTTF on the Two Dimensional RAID with 120 Disks and 20 Strings

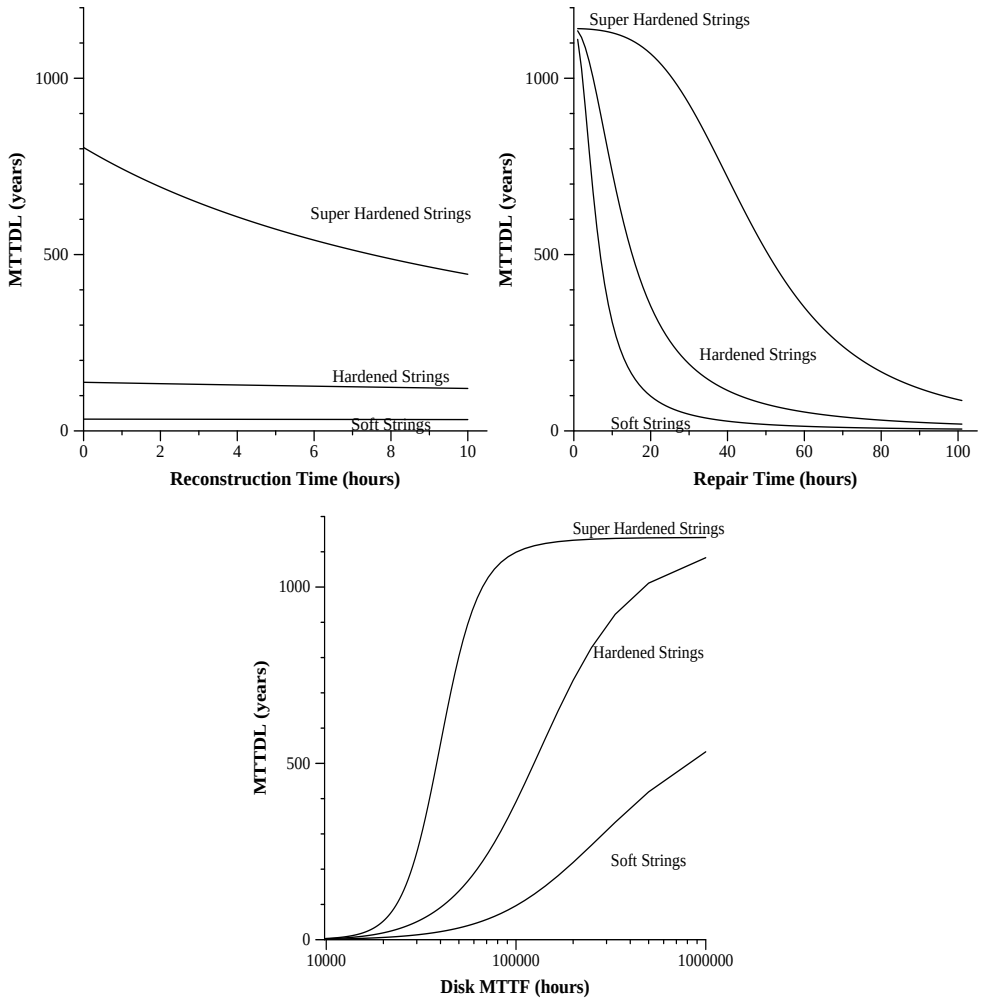


Figure 5.16: Impact of Reconstruction Time, Repair Time and Disk MTTF on the Two Dimensional RAID with 120 Disks and 20 Strings with SDS (1 Disk)

Chapter 6

Addressing in Detail

6.1 Address Translation Schemes

The busiest device limits performance in a RAID. In the naive RAID 4 organization, check disks will become hot spots during a write burst: Each write request to a disk in a reliability group will have to read and then write the check disks. As long as the ratio of check disks to information disks does not mirror the resulting loads, either check or information disks will become a bottleneck. As we have seen, RAID 5 solves the problem by distributing the amount of check and information data and thus the load equally among disks.

Once this bottleneck has been removed (in the RAID 5 organization) reconfiguration of failed disks causes another variety of hot spot. Reads to the failed disk result in reads to all the disks in the reliability group and double the read load there. We have investigated RAID schemes that use address translation to place reliability groups over all disks. As we have seen, there is a noticeable decrease in reliability as measured in data loss. Of course, the chance of losing a single small file are not dependent on the placement scheme.

We can view the RAID 5 solution as the introduction of an abstraction layer and then distinguish between two conceptual levels or views of the disk farm, a "logical" and a "physical" one. A logical address consists of a string address, a disk in string address, a track address, and a block in track address to determine the location of a block. This block address is then *translated* to a physical block address. The left portion of Figure 6.1

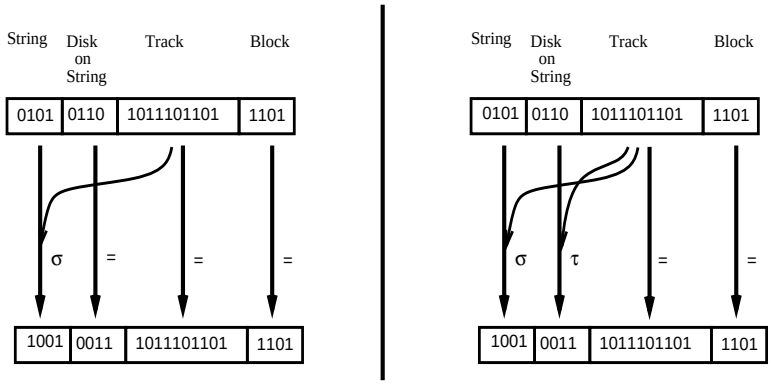


Figure 6.1: Two Address Translation Schemes: RAID 5 (left) and Almost Complete Address Translation.

presents the Level 5 addressing scheme. Based on the track address, the string address is changed by adding the track address modulo n to it. This moves the location of the check information cyclically to the right. The advantage of the RAID 5 addressing scheme is its simplicity. In the literature it is not interpreted as an address translation scheme.

The right portion of Figure 6.1 presents the Almost Complete Addressing Translation Scheme (ACATS). Depending on the track address, the string and the disk in string addresses are permuted. In the next sections we will discuss implementations of this scheme. The Complete Addressing Translation Scheme unifies the string and the disk in string address to a disk address and permutes disk addresses in dependence on the track address. The two schemes differ in their sensitivity to string placements.

6.2 Pseudo-Random Permutations

In this section we investigate one implementation scheme for the Almost Complete Address Translation. In general, ACATS can be based on a table look-up as is used for the mapping of virtual to actual addresses in Virtual Memory implementations. As for these page tables, the size of the table can be prohibitive. For our sample RAID, the table can easily take up more than 100 KBytes. We can compress the table by using cyclic permutations: The string permutation table entry applies to n (the number of strings) consecutive track numbers. All in-string permutations τ are fixed, but the string permutation σ is

followed by a cyclic permutation, which adds a constant i (with $0 \leq i < n$) to the string number modulo n. This scheme reduces the size of the table to $1/n$ of the original size.

We measure the goodness of an ACATS implementation by the frequency in which the same track on two arbitrary disks are grouped in the same reliability groups. In a perfect scheme, this frequency is independent of the two disks, in a good one, this is approximated very closely.

Our alternative to a table look-up, uses a simple algebraic formula like the ones generating the cyclic permutations used for the address translation in RAID 5. The drawback to this approach is the difficulty to find a good formula. As the number of tracks (500 - 2500) is far smaller than the number of possible assignments, formulae that just enumerate are of little use.

Our approach is very similar to the one used in very fast random number generators. When speed is of greater importance than goodness, most random number generators use the "linear congruence method." In these methods, the next random number r_{n+1} is calculated from the previous random number r_n by means of a linear congruence:

$$r_{n+1} = (\text{multiplier } * r_n + \text{addend})\% \text{ modulus}$$

where the multiplier, the addend and the modulus are We will use even simpler generators of the form:

$$r_n = (\text{multiplier } * n + \text{addend})\% \text{ modulus}$$

We only have one arithmetic statement to execute to obtain the n^{th} random number. Of course, a sequence from these kinds of generators will fail almost every randomness test, including that of not working really well for us.

We now convert the random number into a permutation of the strings and the disks inside a string. The basic idea is to use a numeric encoding of a permutation of $\{1, 2, \ldots, k\}$ that assigns each permutation a unique number between 1 and $k!$. We then pick the permutation by its encoding.

Numbering schemes for permutations are well known. We present one due to Knuth ([18]) in Figure 6.2.

6.3 Evaluation of Pseudo-Random Number Based Schemes

Experimental results suggests that the pseudo-random number based addressing schemes work very well for one step translation. Using the scheme in two step translation, where we first permute the string addresses and then the disk in string addresses is more difficult. If there are only 5 disks in a string, there are only 120 permutations of these 5 disks. This and the simple structure of our number generator cause dependencies among the permutations to become visible. We give an example in Figure 6.3: Certain disks on strings 5 and 10 will never be in the same reliability group as disk (0,0). The underlying RAID has 11 strings with 5 disks each. The addressing scheme only uses the cyclic permutation

$$i \mapsto (i + t) \pmod{11}$$

with track number t for the permutation of the strings. This assures us that every string will be the string of checks with equal probability. However, it also brings out dependencies among random numbers, which are obtained using our simple multiplication scheme. The disk 4 will never be in the same reliability group than disk (0,0) for strings 5 and 10.

We can overcome this difficulty by generating the random permutation of disks in a given string out of order for the same track number. The result of one such choice is shown in Figure 6.3.

The best solution is however to use one random number generator that assigns all disks permutations at the same time. The values in Figure 6.3 are obtained by obtaining one long string for each track number, (by multiplying the track number with powers of large primes) that is then translated into disk-in-string permutations. This solves the problem of low-bit dependencies of the random numbers that shows up in terms of disks being much more or much less frequently grouped in the same reliability group.

The example in Figure 6.3 is chosen without search for the most suited parameters used inside the random number generator. Given the dimension of the RAID, we would start a lengthy search to find those that show consistent good spread of reliability groups for all physical disks.

```
void generatePerm(int f,int p[n])
{       for(r=1;r<=n;r++) p[r] = r;
        for(r=2;r<=n;r++)
        {  s = f mod r;
           f = f/r;
           swap(p[r],p[s]);  }
}
```

Figure 6.2: Knuth's Enumeration Algorithm in Pseudo-Code generates Permutation p from integer f. The permutation is an integer array whose indices range from 1 to n.

```
[ 0][0]1000 [ 0][1]  0 [ 0][2]  0 [ 0][3]  0 [ 0][4]  0
[ 1][0] 200 [ 1][1] 75 [ 1][2]150 [ 1][3]300 [ 1][4]275
[ 2][0] 200 [ 2][1]250 [ 2][2]250 [ 2][3]100 [ 2][4]200
[ 3][0] 150 [ 3][1] 25 [ 3][2]225 [ 3][3]350 [ 3][4]250
[ 4][0] 125 [ 4][1]200 [ 4][2]225 [ 4][3]175 [ 4][4]275
[ 5][0] 125 [ 5][1]325 [ 5][2]225 [ 5][3]325 [ 5][4]  0
[ 6][0] 175 [ 6][1]225 [ 6][2]150 [ 6][3]175 [ 6][4]275
[ 7][0] 175 [ 7][1]450 [ 7][2] 75 [ 7][3]100 [ 7][4]200
[ 8][0] 100 [ 8][1]250 [ 8][2]175 [ 8][3]225 [ 8][4]250
[ 9][0] 250 [ 9][1]175 [ 9][2]225 [ 9][3] 75 [ 9][4]275
[10][0] 125 [10][1]125 [10][2]375 [10][3]375 [10][4]  0
```

Figure 6.3: Frequency of Physical Disks being in the same Reliability Group as Disk [0][0]. The underlying RAID consists of 11 strings with 5 disks. The scheme uses one random number generator to generate the disk in string permutation.

```
[ 0][0]1000 [ 0][1]   0 [ 0][2]   0 [ 0][3]   0 [ 0][4]   0
[ 1][0] 210 [ 1][1] 168 [ 1][2] 189 [ 1][3] 191 [ 1][4] 242
[ 2][0] 134 [ 2][1] 218 [ 2][2] 184 [ 2][3] 235 [ 2][4] 229
[ 3][0] 259 [ 3][1] 166 [ 3][2] 207 [ 3][3] 244 [ 3][4] 124
[ 4][0] 191 [ 4][1] 179 [ 4][2] 249 [ 4][3] 126 [ 4][4] 255
[ 5][0] 142 [ 5][1] 205 [ 5][2] 167 [ 5][3] 259 [ 5][4] 227
[ 6][0] 141 [ 6][1] 207 [ 6][2] 165 [ 6][3] 259 [ 6][4] 228
[ 7][0] 194 [ 7][1] 157 [ 7][2] 252 [ 7][3] 125 [ 7][4] 272
[ 8][0] 272 [ 8][1] 165 [ 8][2] 210 [ 8][3] 232 [ 8][4] 121
[ 9][0] 139 [ 9][1] 219 [ 9][2] 188 [ 9][3] 226 [ 9][4] 228
[10][0] 205 [10][1] 151 [10][2] 186 [10][3] 196 [10][4] 262
```

Figure 6.4: Frequency of Physical Disks being in the same Reliability Group as Disk [0][0]. The underlying RAID consists of 11 strings with 5 disks. We use now a different permutation of strings.

```
[ 0][0]1000 [ 0][1]   0 [ 0][2]   0 [ 0][3]   0 [ 0][4]   0
[ 1][0] 200 [ 1][1] 209 [ 1][2] 196 [ 1][3] 202 [ 1][4] 193
[ 2][0] 194 [ 2][1] 183 [ 2][2] 189 [ 2][3] 246 [ 2][4] 188
[ 3][0] 197 [ 3][1] 197 [ 3][2] 215 [ 3][3] 198 [ 3][4] 193
[ 4][0] 196 [ 4][1] 192 [ 4][2] 215 [ 4][3] 199 [ 4][4] 198
[ 5][0] 198 [ 5][1] 189 [ 5][2] 200 [ 5][3] 198 [ 5][4] 215
[ 6][0] 214 [ 6][1] 203 [ 6][2] 215 [ 6][3] 192 [ 6][4] 176
[ 7][0] 165 [ 7][1] 211 [ 7][2] 190 [ 7][3] 237 [ 7][4] 197
[ 8][0] 208 [ 8][1] 199 [ 8][2] 194 [ 8][3] 212 [ 8][4] 187
[ 9][0] 192 [ 9][1] 199 [ 9][2] 198 [ 9][3] 205 [ 9][4] 206
[10][0] 203 [10][1] 206 [10][2] 193 [10][3] 203 [10][4] 195
```

Figure 6.5: Frequency of Physical Disks being in the same Reliability Group as Disk [0][0]. The underlying RAID consists of 11 strings with 5 disks. This scheme uses one random number generator to generate all disk in string permutation at the same time.

Chapter 7

Performance Modeling of Write Synchronization Schemes

We investigate the performance of RAIDs under a load of small read and write accesses, typically to a single block at a time. An analytic treatment of a more realistic load is difficult.

Because an exact queuing model is not feasible, we use a "Queuing Network Model" [20]. Queuing Network Analysis represents a computer system or subsystem as a network of queues. It makes some simplifying assumptions to form and evaluate the resulting analytical model, which yields slightly optimistic approximations. It has built a tradition of extremely valuable results.

While the read operation in a RAID can proceed under normal circumstances as in any disk farm, the write operation is more complicated. To calculate the new check data during a disk write we can either access all disks in the reliability group(s) or we can calculate the data from the difference between the new and the old information data and the old check information. The first procedure involves many more disk operations and will only be performed if we write to all disks in the same reliability group, that is, during a large write in a RAID that involves striping. A naive scheme would read the old information data and then write the new information data while initializing the read of the check disk at the same time. Then the new check data is calculated and after one rotation written to the check disk. This naive procedure actually endangers data on other disks in the reliability

group. A scenario for this has the RAID lose the new check data before it is being written (e.g. in a system crash.) Then any loss of information disks in the reliability group leads to data loss.

We can deal with this danger using a number of strategies.

1. We can reduce the time window of vulnerability between the write of the information disk and the write of the check disk as far as possible, that is, within a full rotation time by synchronizing all necessary writes. A write request would acquire and keep a hold on all disks involved until all writes have been completed. We will see, that this *strong synchronization scheme* has rather disappointing performance figures.

2. Instead of allowing a write request to lock up needed disks, we can use *write-restart* synchronization, in which a write request fails provisionally, when one of the needed disks is busy and is after a short time restarted, until it finally succeeds.

3. We can use a non-volatile cache memory that stores write requests and intermediate data safely.

4. We can ignore the danger.

We treat failures as unrelated, which makes an argument for the last alternative, but certain failure modes, e.g. those linked to power surges, will likely affect several components.

We treat two write synchronization schemes, in Chapter 8 and in Chapter 9, respectively. We discuss the performance without synchronization in Chapter 10. The results are valid over a broad range of RAID organizations. Only when we discuss the effects of failed components on the performance and especially during the data reconstruction process, in Chapter 11, do we observe a strong dependence on the RAID organization.

Chapter 8

Performance with Strong Synchronization

We investigate here the performance impact of a strong write synchronization scheme. The bad performance numbers that we obtain indicate the need for a non-volatile cache.

We assume a disk array with a large number of disks. We also assume a system of load distribution that distributes the load quantitatively and qualitatively uniformly through the array. For a classic Level 5 RAID we would consider only one reliability group at a time and thus treat the RAID as consisting of m (the number of disks per string) distinct storage units. For a RAID that uses complete or almost complete address translation we use a model that encompasses the whole array. Our model is more exact in the latter situation.

8.1 The Model

We assume that loads at individual disks are essentially identical and consist of single block accesses. We model the service times for reads and writes with exponential distributions. We assume the request arrivals to be Poisson distributed. This leads to reasonably conservative performance bounds on the system. Furthermore, we assume that the disks are not synchronized.

A read operation is very straightforward provided no failure has occurred. A single information disk is accessed. We discuss operations under failure in Chapter 10.

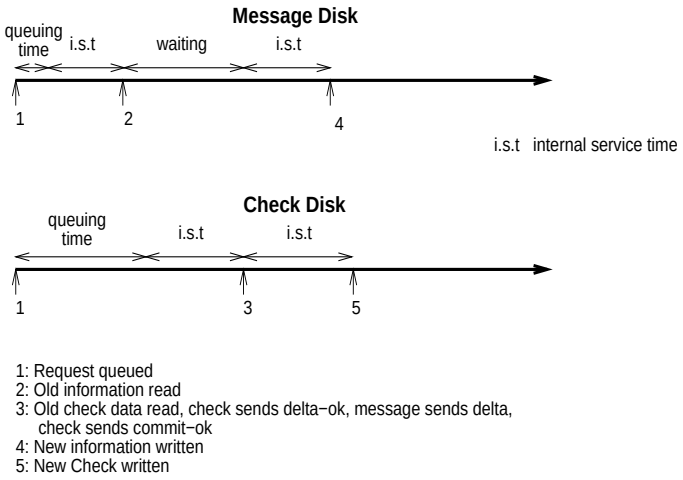


Figure 8.1: Timing Diagram for an Update Operation in the Strong Synchronization Scheme

We now discuss the update operation. A single write operation has to update check information as well. If only a single block is updated at a time, the best stratagem is to use the difference between the new and the old information data (which is stored at the "message disk") to calculate the new check information from the old check information. The write update requires a read and a write at the message disk and all the check disks. The strong synchronization scheme requires the writes to be performed as much as possible in lock step. As we assumed that the disks are not synchronized, we cannot even assume that the rotation phases are finished at the same time. Message and check disks communicate with signals. When a check disk has read the old check information, it sends a Δ-*ok* message to the information disk. This signals that it is ready to receive the difference between old and new information data, the Δ. When the message disk has calculated the Δ and when it has received all *Delta-ok* messages, it sends the Δ value to the check disk. When all check disks have sent their commit-ok to the message disk, the message disk will write the new information. At the same time, the check disks will commit. Hence, the protocol has all necessary changes made at almost the same time. In the worst case, the disk writes will be spread out over one full rotation time of the disk (2 latencies.) Our scheme makes check or message disks likely to wait for the other disks to catch up. We call this time the *"synchronization time"* to distinguish it from the waiting time, as this term is used

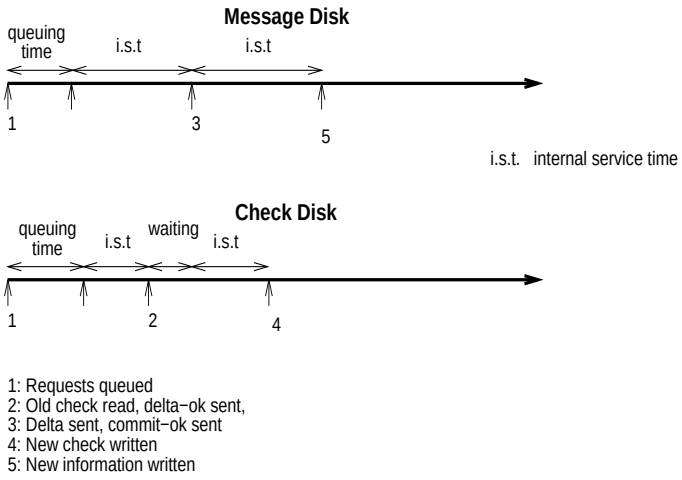


Figure 8.2: Timing Diagram for an Update Operation in the Strong Synchronization Scheme

in queuing theory. (There it refers to the time between the entrance of a request in a server queue and the execution of service; some authors use "queuing time" instead.) The synchronization time represents the cost of synchronization for the disk write operations.

We illustrate the scheme for a single check disk with two timing diagrams. The first is in Figure 8.1. At time 1 a write request is entered in the queue at both the message and the check disk. In this example, both requests have to wait for the disk to be free. We refer to this time as the queuing time. The message disk is free first. A seek phase ensues which is part of the "internal service time" (i.s.t.). At time 2 the old information is read and stored in buffer at the disk. The message disk enters the waiting phase. When the check disk is free, it starts its seek phase. When the old check data is read, it sends the Δ-*ok* message to the message disk, which responds by sending the Δ between new and old information data. The check disk acknowledges with the *commit-ok* signal. The information disk now writes the new information. The time between the reception of the *commit-ok* signal and the actual writing (4) is the time the disk needs to rotate the block again under the head and is the other part of the internal service time; it can be no longer than two latencies and is on average one. The check disk will also write its data, but it needs exactly 2 latencies to do so. As we have seen, the internal service time consists of the seek and rotation times, that would become necessary at the disks without any synchronization.

Figure 8.2 shows the timing sequence, if the head at the check disk is the first to reach the block that is updated. At time 2, the old check data has been read and the Δ-*ok* signal is sent to the message disk. When the message disk reaches the block (3) it commences the transmission of the Δ value to the check disk. The check responds with the *commit-ok* signal. Now both disks will write the next time that the block appears under the head, that will take 2 latencies for the message disk and possibly less for the check disk.

8.2 Results

Our analytic model, which is presented in Section 8.3, and our simulation results presented in Section 8.9 both show rather disappointing performance. With increasing load, the synchronization time becomes larger. This sets off a chain reaction: The waiting time for requests at disks increases and drives the synchronization time higher. If the load continues to increase, the system becomes unstable. Both our analytic model and the simulation results indicate a catastrophic behavior, in which the performance does not worsen continually with increased load but just exhibits an instability. Interestingly, the utilization of disks at this point falls well short of one, which is another indication of the unacceptable behavior of the system. The maximum tolerable load consisting of writes only is 35.34% of the theoretically possible one if one check disk is used and of 31.83% if two check disks are used.

Our results are applicable to a larger class of queuing problems, where a request needs similar service from two or several servers at the same time. These results are a variety of the fork-join results.

We have corroborated our theory of the system behavior by simulation for both exponentially distributed service times and by using a different service time (constant + uniform service time). In addition we derive a lower bound that is independent of the service time distribution (Section 8.4.) In each case an instability of the system is observed.

Our model's weakness is the somewhat unrealistic assumptions on the load. A more realistic model would include at least directory reads before some writes. Our model instead predicts performance for an extreme case never attained in practise and derives its value from precisely this point. The use of the exponential (i.e. memory-less) distribution in

our model explains the differences between our results and the simulation results. The small magnitude of the difference justifies our approach. Our model assumes a large number of disks. It applies to load balanced Level 5 RAIDs as well as to classic Level 5 RAID schemes, but is more accurate for the former.

8.3 A Queueing Network Approach

8.3.1 Queuing Networks

Queueing network modeling represents a system as a separable network of queues which is then evaluated analytically. This evaluation is based on simplifying assumptions. Queueing theory tends to consider a complicated situation at a single server. Queueing network methodology instead considers many servers with simple characteristics.

We consider requests entering a queue at a server, waiting their time and finally being served. The time of actual service is called the service time. The response time is the time from entrance in the queue to the termination of service; it consists of service time and waiting time. The portion of time, during which the server is busy, is the utilization. The load or arrival rate is the average number of requests arriving during a time interval. We make the *flow balance assumption* which states that load equals throughput, the average number of requests served. If the flow balance assumption is violated, the server receives more requests than it can handle.

8.3.2 Notation

We adjust the general queuing network terminology to our purposes: We call the time, in which the disks are busy actually servicing the request the *internal service time.* This time excludes the time spent waiting for the other disk(s) to react, which we call the *synchronization time.* We group the synchronization time and the internal service time together under the name *external service time.* The external service time can be interpreted as the time an outside observer would find a disk busy with a single request.

The *external response time* is the time from the placement of the request in queue until the disk is released. The *internal response time* is the external response time minus the synchronization time.

Name	Symbol	Short Description
Internal Service Time	D'	Service Time as seen by client
External Service Time	D	Service Time as seen by other client following in line
Internal Response Time	R'	Time until no further operation take place at disk in the equivalent scenario
External Response Time	R	Time until disks are released from start of request

Table 8.1: Notation for the Queuing Network Analysis of the Strong Synchronization Scheme

For modeling purposes we use an *equivalent scenario.* In it the synchronization time is moved (conceptually) behind the last piece of the internal service time. Then it appears that requests are serviced independently at the message and the check disks, but that they do not release the disk until the last one is finished.

Table 8.1 repeats the notation and names the quantities for our analysis in tabular form.

8.3.3 A Queueing Network Approximation for One Check Disk

Before we give our general result, we introduce our approach for a RAID handling only write requests and with one check disk per reliability group. We designate the internal service time at a disk (seek time, 3 rotation times and transmission time) with D'. We denote with λ the load at an individual disk. The disk has a utilization $U = \lambda D$. With probability $U^{\nu}(1-U)$ there are exactly ν queued clients ahead of any arriving client. Each of these clients will require D time to leave the disk and the same is true for the current

request. Accordingly, the external response time is given as

$$\begin{aligned} R &= D + D(1-U)\sum_{\nu=1}^{\infty} \nu U^{\nu} \\ &= \frac{D}{1-U} \\ &= \frac{D}{1-\lambda D} \end{aligned}$$

The internal response time is the difference of the external response time and the synchronization time, which is the difference between the internal and external service time:

$$R' = R - D + D'.$$

The synchronization time is the difference between the internal response times at the two disks. As we assume all response times to be exponentially distributed, this time is equal to R'. Only one of the disks suffers a synchronization time delay. This implies for the external service time on average

$$\begin{aligned} D &= 1 + (1/2)R' \\ &= 1 + 0.5(R - D + D') \end{aligned}$$

The two equations imply the quadratic equation

$$3\lambda D^2 - (2 + 3\lambda D')D + 3D' = 0.$$

We choose the correct root by letting $\lambda \longrightarrow 0$ and consequentially $D \longrightarrow 1$ and obtain

$$D = \frac{(2+3\lambda) - \sqrt{4 - 24\lambda + 9\lambda^2 D'}}{6\lambda}$$

The solution D exists only if $\lambda < \frac{24-\sqrt{432}}{18} = .1787$ The maximum possible load without synchronization costs would have been $\lambda = 0.5$. Our model sets the external service time at 1.5 and the synchronization time at 0.5 for zero load. Beyond the maximum feasible load the system becomes instable and cannot keep up with arriving requests. We give the numerical results in Figure 8.3.

8.3.4 The Queuing Network Approach in the General Case

To extend our approach to more than one set of writes we need to calculate the synchronization time among several disks. We can show that the *average synchronization*

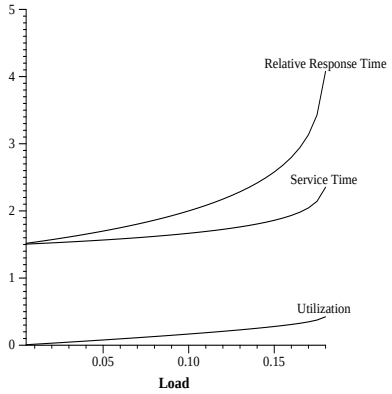


Figure 8.3: Service Time, Response Time and Utilization for Strong Write Synchronization (1 Check and Writes Only Load)

time among x different writes is equal to $H_x - 1$, where H_x is the x^{th} harmonic number. We prove the formula in Section 8.7.

We obtain the response times for read and write operations as response times within a two-population queue in our queuing network. The external service times are D_r and D_w respectively. An individual disk sees write and read loads λ_w and λ_r. The disk experiences utilization $U = \lambda_w D_w + \lambda_r D_r$. With probability $U^\nu(1-U)$ an arriving request finds ν clients ahead of it. The device is busy with probability $u_r = \lambda_r D_r$ with a read client and with probability $u_w = \lambda_w D_w$ with a write client. An unknown client will require external service time $\overline{D} = \frac{u_r}{U} D_r + \frac{u_w}{U} D_w$. Accordingly, the read external response time is given by

$$\begin{aligned} R_r &= D_r + \overline{D}(1-U)\sum_{\nu=1}^{\infty} \nu U^\nu \\ &= D_r + \overline{D}\frac{U}{1-U} \\ &= \frac{D_r + \lambda_w D_w (D_w - D_r)}{1-U}. \end{aligned}$$

We obtain a similar equation for the write external response time:

$$R_w = \frac{D_w + \lambda_r D_r (D_r - D_w)}{1-U}.$$

Now we determine the external service times. Since read operations do not involve both a read and a consecutive write and as they involve only one disk, the external read service

time coincides with the internal read service time and is a fraction α of the internal write service time I. We are left with the more interesting problem of determining the write service time. The result in Section 8.7 gives us

$$(H_x - 1)(R_w - D_w + I)$$

for the synchronization time. Accordingly, the external service times are given by

$$\begin{aligned} D_r &= \alpha I \\ D_w &= H_x I + (H_x - 1)(R_w - D_w) \end{aligned}$$

The two expressions for D_w derived in this section lead to a quadratic equation, from which D_w and then D_r as well as all the other variables can be determined. The quadratic equation does not have a solution if the load and the write proportion of the load exceed a relatively small fraction of the theoretically possible values.

8.3.5 An Example

We assume an actual average seek time of 6 ms (due to locality of disk references) and a latency of 6 ms. We neglect the transfer time and the controller overhead. The internal write service time is $I = 24$ms and the (internal = external) read service time is $D_r = 12$ms. We assume that the RAID has only one check disk per reliability group and hence uses two disk writes for a single update operation. We use a read to write relation of 3:1, that is, 1/4 of all arriving requests at the RAID are writes. Let Λ be the request arrival rate at the RAID which has N disks. Each write request generates 2 individual write requests at a disk, and the disk loads are

$$\lambda_w = \frac{\Lambda}{2N} \quad ; \qquad \lambda_r = \frac{3\Lambda}{4N}.$$

We abbreviate $\lambda = \Lambda/N$. We obtain the external write service time

$$D_w = \frac{(2 + 18\lambda) - \sqrt{3564\lambda^2 - 360\lambda + 4}}{3\lambda}$$

The equation is solvable only for $\lambda < 0.012710531.$ or $\Lambda < N \cdot 0.012710531$. Without synchronization the disk utilization is $0.5 \cdot \lambda \cdot 24 + .75 \cdot \lambda \cdot 12 = 21\lambda$ and the maximum tolerable

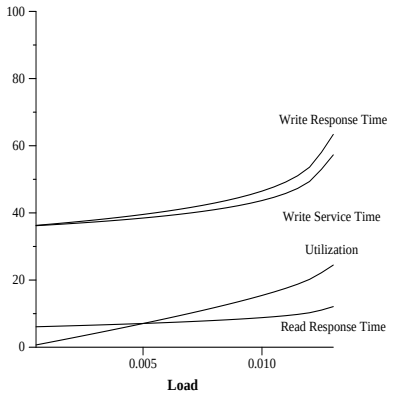


Figure 8.4: Service Time, Response Time and Utilization for Strong Write Synchronization (1 Check and a Writes Only Load)

RAID request per disk is $\lambda = 0.047619048$. We observe an interesting phenomenon if we calculate D_w for the boundary case $\lambda = 0.012710531$. Then $D_w = 58.449946164$ and the utilization at each disk is only $U = 0.371464926$. Accordingly, the system will become rather suddenly unstable, as the load passes over the critical load value. The one server queue does not show such a pathological behavior and experiences a more graceful degradation. We present the development in Figure 8.4. The figure does not represent the final knee well.

8.4 A General Lower Bound for Synchronization Schemes

We have observed that a synchronization scheme between disks renders the system instable. We now show, that this behavior is general and does not depend on the service time distribution used. We use a constant service time distribution which will minimize both queue length on arrival and the synchronization time at load zero. We still assume that requests arrive in a memoryless way. the queuing theoretical literature calls this kind of queue an M/D/1 queue (for Markov arrival and Deterministic service time at 1 server.)

Let p_i stand for the probability of i new requests arriving at the same time:

$$p_i = \frac{\mu^i}{i!} e^{-\mu}.$$

We denote with π_i the probability that at any given time there are i requests in queue. As the arrivals are the result of a Poisson process an arriving request sees the same number

Load	$\mathcal{S}$	Lower Bound
0.000000	0.000000	1.000000
0.050000	0.048814	1.051319
0.100000	0.095527	1.105616
0.150000	0.140580	1.163575
0.200000	0.184473	1.226201
0.250000	0.227781	1.294970
0.300000	0.271185	1.372090
0.350000	0.315510	1.460941
0.400000	0.361790	1.566884
0.450000	0.411375	1.698874
0.500000	0.466085	1.872956
0.550000	0.528492	2.120854
0.600000	0.602402	2.515102
0.650000	0.693751	3.265320
0.700000	0.812409	5.330754

Table 8.2: Lower Bounds for External Service Times in a System with a Large Number of Devices. Each requests requires synchronized service at two devices. The internal service time is constant 1. For loads of 3/4 or more, the system is shown to be unstable.

of requests queued as an outside observer. We use conditional probabilities and that the busy time is$1 - \mu D$ independent of the service distribution to derive a well known recursive formula:

$$\pi_i = p_i\pi_0 + p_i\pi_1 + p_{i-1}\pi_2 + \ldots + p_1\pi_i + p_0\pi_{i+1}.$$

We calculate numerically the probabilities π_i. We give a first estimate of the expected synchronization delay as the expected difference in waiting time between two customers entering two different queues. The service time is deterministic 1 and so the waiting time

equals the number of customers:

$$\mathcal{S} = \sum_{k=0}^{\infty} \pi_k \sum_{\mu=l}^{\infty} (l-k)\pi_l.$$

We add the waiting time and obtain the expected value for the external service time in first approximation. We now calculate the additional waiting time resulting from the increase in external service time. Our approximation for the external service time depends on the number of customers at other disks and is no longer deterministic. It also does not include the waiting time we just calculated. If we assume that the first approximation is deterministic, we will underestimate further increases in the waiting time. Because $\mathcal{S}$ is the proportional increase in service time, we calculate the external service time D by an infinite repetition of the previous step as

$$D = \sum_{\nu=0}^{\infty} \mathcal{S}^{\nu} = \frac{1}{1-\mathcal{S}}$$

The results are presented in Table 8.4. Above 75% of the nominally possible load renders the system unstable.

This Section shows that a strong synchronization scheme with an infinite number of servers (e.g. disks in a RAID) with the best possible service time distribution (constant service time) will not be able to reach full utilization before the queuing network becomes unstable. The infinite number of servers model approximates the situation very well for a moderate and larger set of servers. Only for few servers is this approximation bad. If there are only two servers however, the approximation becomes ludicrously bad, we are then talking about mirrored disks for which the strong synchronization only means that all operations are performed in tandem. Apart from this case, the result is strong evidence for the accuracy of the queuing network predictions.

8.5 Performance with Synchronized Disks

8.5.1 Results

RAID systems can use disk synchronization to keep disks rotating strictly in parallel. Then the overwrites of an update operation perform at the same time. The performance is improved somewhat by this change.

8.5.2 Derivation

We model the new situation by assuming that only the seek and the first rotation phase I_0 of an update operation are exponentially distributed. The second rotation phase I_1 is uniformly distributed and the same at all disks. We obtain the internal write service time I as the sum of I_0 and I_1. We assume that the internal response time R' excluding I_1 is exponentially distributed. We obtain the synchronization time by applying our result of Section 8.7 to the modified internal response time. The new formula for the external write service time reads now:

$$\begin{aligned} D_w &= (H_x - 1)(R' - I_1) + I_0 + I_1 \\ &= (H_x - 1)(R_w - D_w) + H_x I_0 + I_1 \end{aligned}$$

The other formulae from Section 8.3.4 are still valid:

$$\begin{aligned} R_r &= \frac{D_r + \lambda_w D_w (D_w - D_r)}{1 - U} \\ R_w &= \frac{D_w + \lambda_r D_r (D_r - D_w)}{1 - U} \\ D_r &= I_0 \\ U &= \lambda_r D_r + \lambda_w D_w. \end{aligned}$$

We can solve the system algebraically and obtain a quadratic equation for D_w.

8.5.3 An Example

We modify the example in Section 8.3.5. With the same notation and under the same assumptions we have:

$$\begin{aligned} \lambda_w &= \frac{\lambda}{2} \\ \lambda_r &= \frac{3\lambda}{4} \\ D_r &= I_0 = 12ms \\ I_1 &= 12ms \end{aligned}$$

The maximum load is now $\lambda = 0.014720830$ which represents a 16% increase in the maximum load but is still only 31% of the maximum load without synchronization.

8.6 Performance of Level 4 RAIDs with Strong Write Synchronization

In this section we consider the performance of Level 4 RAIDs under a Poisson load of small reads and writes. We use the same locking scheme as before: an update operation holds all needed disks and finishes within a full disk rotation.

In each reliability group one (or more) disk(s) contain(s) only check information and is accessed in each update operation.

Our model is similar to the one presented for Level 5 RAIDs. We now have two classes of servers, the message and the check disks. We distinguish all magnitudes by indices I and C, if necessary. We give the formulae in a form which incorporates read loads at check disks, even though normally these are zero.

$$\begin{aligned} R_{r,I} &= \frac{D_r + \lambda_{w,I} D_{w,I}(D_{w,I} - D_r)}{1 - U_I} \\ R_{r,C} &= \frac{D_r + \lambda_{w,C} D_{w,C}(D_{w,C} - D_r)}{1 - U_C} \\ R_{w,I} &= \frac{D_{w,I} + \lambda_{r,I} D_{r,I}(D_{r,I} - D_{w,I})}{1 - U_I} \\ R_{w,C} &= \frac{D_{w,C} + \lambda_{r,C} D_{r,C}(D_{r,C} - D_{w,C})}{1 - U_C} \end{aligned}$$

The read service time D_r is always constant. The write service time includes the synchronization time. The write response times will differ between message and check disks because the loads and the utilization do. If I denotes the internal write service time, then

$$\begin{aligned} D_{w,I} &= I + \mathcal{S}_{\mathcal{I}}(R_{w,I} - Dw, I + I, R_{w,C} - Dw, C + I) \\ D_{w,C} &= I + \mathcal{S}_{\mathcal{C}}(R_{w,I} - Dw, I + I, R_{w,C} - Dw, C + I) \end{aligned}$$

where the synchronization times $\mathcal{S}_{\mathcal{I}}$ and $\mathcal{S}_{\mathcal{C}}$ are given and derived in Section 8.8.

Our system of equations is no longer amenable to closed form solution, but can be easily solved numerically by back-substitution. As should be expected, the performance worsens even further if the load at check and message disks differ.

8.7 Synchronization Time for Identical Distributions

We derive the formula of the expected synchronization time among n independently identically exponentially distributed random variables X_i with mean $\frac{1}{\alpha}$.

We had defined the synchronization time as the time between the execution of one request and the last of the $n-1$ other requests. If we interpret the random variables as the finishing times of a race, then the synchronization time is the difference between my time and the time of the last of the other racers to finish. If I am the last, then the synchronization time is zero.

We call $t^+ = \frac{|t|+t}{2}$ the positive part of a real number t, (t^+ is equal to t if t is positive and zero otherwise. We denote the (cumulative) distribution of any of the variables X_i with F, that is,

$$\mathcal{P}(X_i < t) = 1 - F(t) = 1 - e^{-\alpha t}.$$

The density of a single random variable is $f(t) = \alpha e^{-\alpha t}$. The distribution of the maximum of l independently distributed variables is obtained from $\mathcal{P}(X_{(l)} < t) = (1 - e^{-\alpha t})^l$ with density $f_l := l\alpha e^{-\alpha t}(1 - e^{-\alpha t})^{l-1}$. The expected value of the synchronization time is given as a Riemann integral. The synchronization time is the expected value for the difference between the maximum of the l independently distributed variables and a single one.

$$\begin{aligned}
\mathcal{S} &= \int_0^\infty \int_0^\infty (s-t)^+ f_l(s)ds\, f(t)dt \\
&= \int_0^\infty \int_0^s (s-t) f(t)dt\, f_l(s)ds \\
&= \int_0^\infty \left(s + \alpha^{-1}e^{-\alpha s} - \alpha^{-1}\right) f_l(s)ds \\
&= \frac{H_{n-1}}{\alpha} + \frac{1}{n\alpha} - \frac{1}{\alpha} \\
&= \frac{H_n - 1}{\alpha}
\end{aligned}$$

In the calculation we used

$$\begin{aligned}
\int_0^s (s-t)\alpha e^{-\alpha t}dt &= s\int_0^s \alpha e^{-\alpha t}dt - \int_0^s t\alpha e^{-\alpha t}dt \\
&= -s\left[e^{-\alpha t}\right]_{t=0}^{t=s} + \left[(t+\alpha^{-1})e^{-\alpha t}\right]_{t=0}^{t=s} \\
&= s + \alpha^{-1}e^{-\alpha s} - \alpha^{-1}
\end{aligned}$$

$$\begin{aligned}
&\int_0^\infty \alpha^{-1} e^{-\alpha s} \tfrac{d(1-e^{-\alpha s})^{n-1}}{ds} ds \\
&= \left[\alpha^{-1} e^{-\alpha s}(1-e^{-\alpha s})^{n-1}\right]_{s=0}^{s=\infty} + \int_0^\infty (1-e^{-\alpha s})^{n-1} e^{-\alpha s} \\
&= 0 + \frac{1}{n\alpha}\int_0^\infty 1 f_n(s) ds \\
&= 1/(n\alpha)
\end{aligned}$$

and

$$\int_0^\infty s f_{n-1}(s) ds = \frac{H_{n-1}}{\alpha}$$

8.8 Synchronization Time for 2 Classes of Distribution

We develop formulae for the synchronization time for two classes of exponentially distributed random variables. These represent response times at check disks and the response time at an information disk. We only give results for the synchronization between one information and one check disk and between one information and two check disks.

We have two independent exponentially distributed random variables with expectation $\frac{1}{\alpha}$ and one additional independent exponential distributed random variable with expectation $\frac{1}{\beta}$. (These of course represent the reaction time at check and information disks.) The (cumulative) probability distributions are denoted by F and G respectively. We have density

$$f(t) = \alpha e^{-\alpha t}$$

for the first single variable and

$$g(t) = \beta e^{-\beta t}$$

for the second single variable. The maximum of two variables describing check disk response is

$$f_2(t) = 2\alpha e^{-\alpha t}(1 - e^{-\alpha t}).$$

The maximum of a mixed pair of check and message response times has density

$$h(t) = \alpha e^{-\alpha t} + \beta e^{-\beta t} - (\alpha + \beta) e^{(\alpha+\beta)t}.$$

First we calculate the synchronization times for an update operation that involves one check and one message disk. The synchronization time at the message disk is

$$\begin{aligned}
\mathcal{S}_{\mathcal{I}} &= \int_0^\infty \int_0^\infty (s-t)^+ f(s)ds\, g(t)dt \\
&= \int_0^\infty \int_0^s (s-t)g(t)dt\, f(s)ds \\
&= \int_0^\infty (s + \beta^{-1}e^{-\beta s} - \beta^{-1}) f(s)ds \\
&= \frac{1}{\alpha} + \frac{\alpha}{\beta}\frac{1}{\alpha+\beta} - \frac{1}{\beta} \\
&= \frac{\beta}{\alpha(\alpha+\beta)}
\end{aligned}$$

We obtain for the synchronization at the check disk *mutatis mutandis:*

$$\mathcal{S}_{\mathcal{C}} = \frac{\alpha}{\beta(\alpha+\beta)}$$

We now come to the result that applies to an update operation that involves two check and one message disk. The calculation is straight-forward:

For the information disk we have:

$$\begin{aligned}
\mathcal{S}_{\mathcal{I}} &= \int_0^\infty \int_0^\infty (s-t)^+ f_2(s)ds\, g(t)dt \\
&= \int_0^\infty \int_0^s (s-t)g(t)dt\, f_2(s)ds \\
&= \int_0^\infty \left(s + \beta^{-1}e^{-\beta s} - \beta^{-1}\right) f_2(s)ds \\
&= \int_0^\infty s f_2(s)ds + \frac{2\alpha}{\beta}\left(\int_0^\infty e^{-(\alpha+\beta)s}ds - \int_0^\infty e^{-(2\alpha+\beta)s}ds\right) - \frac{1}{\beta} \\
&= \frac{3}{2}\alpha^{-1} + \frac{2\alpha}{\beta}\left(\frac{1}{\alpha+\beta} - \frac{1}{2\alpha+\beta}\right) - \frac{1}{\beta} \\
&=: \mathcal{S}_{\mathcal{I}}(\alpha,\beta)
\end{aligned}$$

The expected synchronization delay at a check disk is

$$\begin{aligned}
\mathcal{S}_{\mathcal{C}} &= \int_0^\infty \int_0^\infty (s-t)^+ h(s)ds f(t)dt \\
&= \int_0^\infty \int_0^s (s-t)f(t)dt h(s)ds \\
&= \int_0^\infty \left(s + \alpha^{-1}e^{-\alpha s} - \alpha^{-1}\right) h(s)ds \\
&= \int_0^\infty s h(s)ds + \int_0^\infty \alpha^{-1}e^{-\alpha s}h(s)ds - \int_0^\infty \alpha^{-1}h(s)ds
\end{aligned}$$

$$
\begin{aligned}
&= \alpha^{-1} + \beta^{-1} - (\alpha+\beta)^{-1} \\
&\quad +(2\alpha)^{-1} + (\alpha+\beta)^{-1} - (\alpha+\beta)(2\alpha^2+\beta)^{-1} \\
&\quad -\beta^{-1} \\
&= : \mathcal{S}_2(\alpha^{-1}, \beta^{-1})
\end{aligned}
$$

8.9 Simulation Results

We have tested our analytical results in a variety of settings and have found that the expected blow-up of the response time happens always for loads shortly after the predicted maximum load. We present the examples in Figures 8.5 to Figure 8.6

We are simulating a RAID with 100 disks. The latency of the disks is 12 ms and the maximum seek time 22 ms. We assume a completely random disk accesses. The load contains a portion of ρ_w write and $\rho_r = 1 - \rho_r$ writes. We give the load in our scatter diagrams in terms of RAID requests per millisecond.

In Figure 8.5 we give the results of several simulation runs with a write quorum of 3 (2 Check Disks per Reliability Group) and a load consisting of writes only. The average service time is 27 ms. Figure 8.6 gives the response times in the same situation with a write quorum of 2.

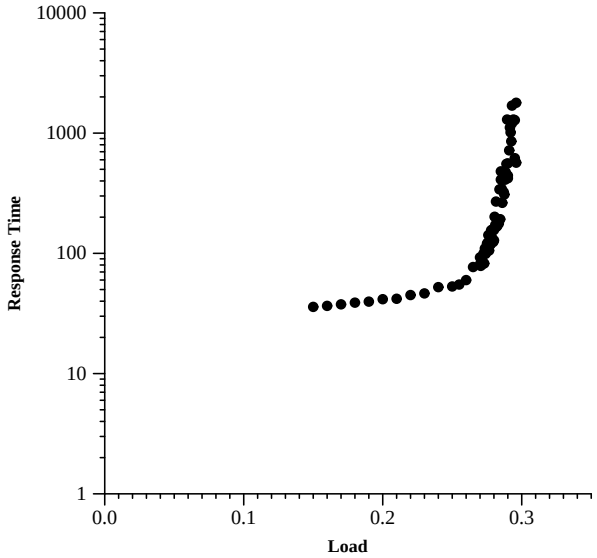


Figure 8.5: Response Time under the Strong Synchronization Scheme: Write Load only, Write Quorum of 3.

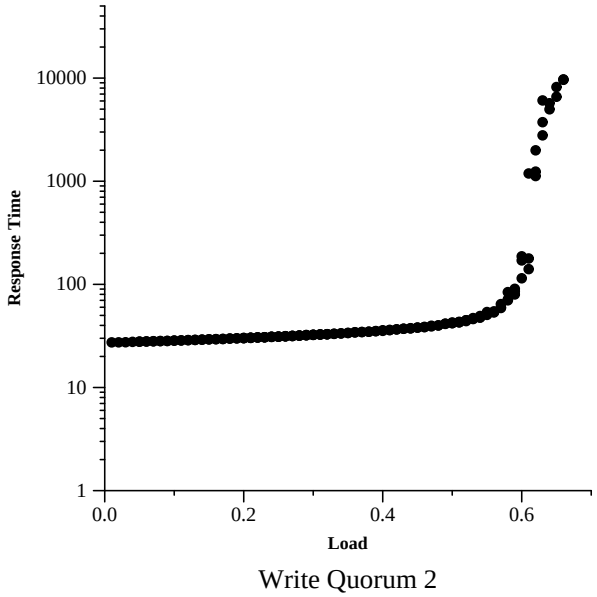


Figure 8.6: Response Time under the Strong Synchronization Scheme: Write Load only, Write Quorum of 2.

Chapter 9

Performance under Write-Restart Synchronization

The disappointing performance figures of strong synchronization arise from the required synchronization among disks participating in an update. The response time reflects the time of the slowest participant not only of the actual request but of unfinished update operations ahead. The write-restart provides much better response times. Here the update operation can proceed only if all participating disks are idle. The updates are not executed in arrival order, but the synchronization scheme does not provide any scheduling optimization. Writes to the same block will be executed in arrival order. There is a small possibility of starvation which can be thwarted within the implementation. Once a logical write operation begins, it continues without interruption to completion.

The scheme applies to Level 5 RAIDs as well as to Level 4 RAIDs.

9.1 Derivation of Results

Our analysis of the response times proceeds much as within Chapter 7 and uses the same notation and assumptions. The external read response time R_r is, as in Section 8.2,

$$R_r = \frac{D_r + \lambda_w D_w (D_w - D_r)}{1 - U}.$$

The read service time D_r is independent of loads and only a fraction of the internal write service time I. Since all disks among an update operation begin together and since we

assume them to be exponentially distributed, the external write service time would be $H_x I$. We can derive a better estimate by modeling only the seek and the initial rotation phase with an exponential distribution. We decompose the internal service time I into components I_0 and I_1 where the first component I_0 consists of the seek and the first rotation phase. After both disks have read the block to which we are writing, the operation terminates with the last disk to write to the block after exactly one full rotation time I_1. Because the two seek phases are not synchronized, it does not matter whether the RAID uses rotational synchronization or not. We obtain a better estimate for the synchronized write service time (involving x disks) and designate it with $F_x D_w$:

$$F_x D_w = H_x I_0 + I_1.$$

The utilization U at a disk consists of the read utilization $u_r = \lambda_r D_r$ and the write utilization $u_w = \lambda_w D_w$. The loads λ_r and λ_w denote the number of respective requests per time unit at an individual disk. The read service time D_r consists only of a seek and a rotation phase and coincides with I_0. A request of unknown nature takes time

$$\overline{D} = (u_r/U)D_r + (u_w/U)D_w.$$

If a write request arrives, it will possibly find both devices idle (with probability $(1-U)^2$) and then experience a response time of $H_2 D_w$. If only one of the devices is busy (which happens with probability $2U(1-U)$) we first wait for that device to become free. This happens after the "stay-busy" time B terminates. At this point, we probe the other device, which in turn is idle with probability $1-U$. If not, we wait for the stay-busy time to be over there and reprobe the first device, *et cetera*, until finally we find both disks to be free. We use the same probing scheme if originally both devices are busy, only then we need to wait for the maximum of the busy times, or approximately $H_2 B$. Our response time formula is

$$\begin{aligned} R_w &= F_2 D_w \\ &\quad +2U(1-U)\left(B + U(B + U(B + \ldots))\right) \\ &\quad +(1-U)^2\left(H_2 B + U(B + U(B + \ldots))\right) \\ &= F_2 D_w \end{aligned}$$

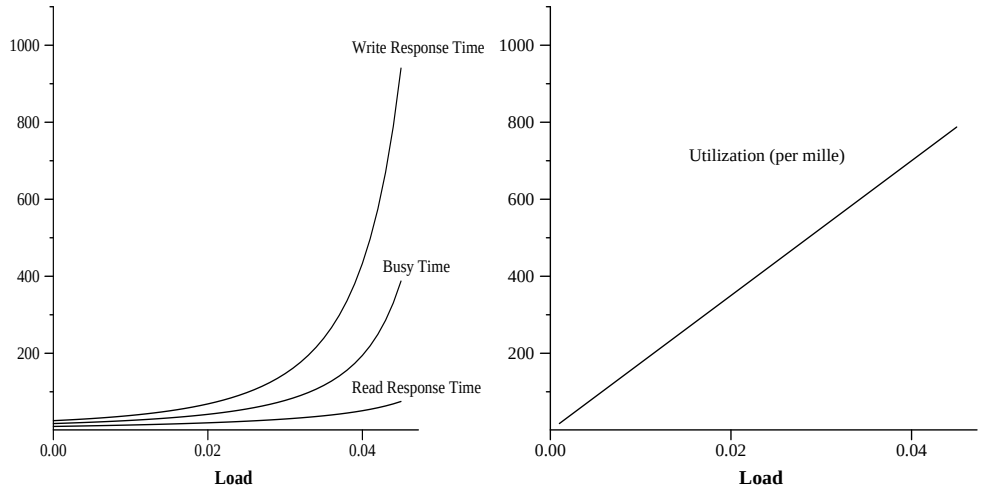


Figure 9.1: Response Times, Busy Times and Utilization (per mille) for the Write Restart Scheme with Two Disks

$$
\begin{aligned}
& +2U(1-U)\cdot B\cdot\sum_{\nu=0}^{\infty}U^{\nu} \\
& +(1-U)^2\cdot(H_2B+\sum_{\nu=1}^{\infty}BU^{\nu}) \\
= \quad & F_2D_w \\
& +2U(1-U)\cdot B\frac{1}{1-U} \\
& +(1-U)^2\cdot(H_2B+\frac{BU}{1-U}) \\
= \quad & F_2D_w+3UB+(1-U)^2H_2B-U^2B
\end{aligned}
$$

We illustrate our model for the write restart in Figure 9.1, where we take the seek time to be 5 milliseconds and the latency to be 5 milliseconds also.

We now amend the formula for writes involving three disks simultaneously. If an arriving write request finds one or more of the needed disks busy, we wait for all of them to be done. Then we check whether the other two are still free. If not, we wait for that disk to be free, and so on. To simplify the formula, we call $p_{i,n}$ the probability that exactly i disks out of n are busy at a given moment. This way we obtain:

$$R_w \quad = \quad F_3D_w$$

$$
\begin{aligned}
&+p_{1,3}B + p_{2,3}H_2B + p_{3,3}H_3B \\
&+(1-p_{0,3})(p_{1,2}B + p_{2,2}H_2B) \\
&+(1-p_{0,3})(1-p_{0,2})(p_{1,2}B + p_{2,2}B) \\
&+ \quad \vdots \\
= \quad & F_3D_w + p_{1,3}B + p_{2,3}H_2B + p_{3,3}H_3B + (1-p_{0,3})\frac{1-p_{0,2}}{p_{0,2}}(p_{1,2}B + p_{2,2}B)
\end{aligned}
$$

We calculate the average length of time for a disk found busy to become idle again. Typically, in the literature, busy time is the time average amount of time a device is busy, when an arriving request finds it busy. The average number of requests at a busy disk is

$$\sum_{\nu=0}^{\infty}(\nu+1)U^{\nu}(1-U) = \frac{1}{(1-U)}.$$

It takes clearing time $\overline{D}/(1-U)$ to clear the disk, but in the mean time $\lambda\overline{D}(1-U)^{-1}$ additional requests have arrived. Summing up the resulting geometric series we arrive at an average busy time of

$$\left(\sum_{\nu=0}^{\infty}\lambda^{\nu}\overline{D}^{\nu}\right)\frac{\overline{D}}{(1-U)} = \frac{\overline{D}}{(1-U)^2}$$

The strong write synchronization scheme, which we investigated in the previous Chapter, exhibited singular behavior. As the load increases over a certain point, suddenly the expected response time jumps from a finite value to infinity. In contrast, the present scheme degrades gracefully as we reach maximum load. The repsonse time graphs have a pole at this point. While the overall performance of write-restart synchronization is still poor, this facet of behavior is acceptable.

Chapter 10

Performance Without Write Synchronization

We have used write synchronization to prevent data inconsistencies after a system crash, which, if not remedied, could destroy the capability of a RAID to regenerate lost data after some component failure. Our two schemes with write synchronization suffered high performance losses. The alternative design uses a non-volatile data cache. This solution is the current state of the art. Our previous results should be interpreted as strong evidence that no alternative to non-volatile cache is feasible.

In Chapter 4 we have proposed to build the non-volatile data buffer from two components, a non-volatile data cache and a BIDA (Balanced Information Dispersal Array) based storage buffer that uses a small portion of the disk space in the RAID. The BIDA storage buffer serves as an overflow for the non-volatile cache and is somewhat faster than the RAID itself.

In our performance analysis, we assume that the temporary storage of requests is not a bottleneck. The analysis is easy and does not depend on all the features of a RAID architecture. We make our analysis for a storage unit of N disks, organized into n strings and m reliability groups, so that $N = nm$. We assume that all N disks are the target of an update request with the same probability. For a RAID with ACATS for load balancing, the storage unit is the whole RAID. (Even though most of our RAID organizations do never pair disks in the same string as message and check disks, the performance calculation does

not depend on that fact.) For a RAID organization which assigns disks to fixed reliability groups, the N disk storage unit is the reliability group. We will discuss RAIDs which use fixed message and check disks further below.

We adopt the same notation as before. An update involves w writes. The system experiences a load of Λ requests, of which a fraction ρ_r are read requests and a fraction ρ_w are updates. The service time D_r for a read request consists of seek time, rotation time (of one disk latency), the transfer time and excludes control time. The service time D_w for a write operation contains in addition two latencies, because message and check data needs to be read and processed first before they can be written. We can safely assume that all processing is done in parallel with disk activities and hence we do not include it in our performance calculations.

The overall load at an individual disk consists of $\lambda_r = \rho_r \Lambda / N$ reads and $\lambda_w = w\rho_w \Lambda / N$. The utilization U at a disk embodies the read utilization $U_r = \lambda_r D_r$ and the write utilization $U_w = \lambda_w D_w$; that is,

$$U = U_r + U_w = (\rho_r D_r + w\rho_w D_w)\Lambda / N.$$

A request of unknown nature is with probability U_r/U a read request and with probability U_w/U a write request and consequentially takes service time

$$\overline{D} = \frac{U_r D_r + U_w D_w}{U}$$

The read response time consists of the read service time and the time it takes for previous requests to clear the device. As the average queue length is given by the geometric series $\sum_{\nu=1}^{\infty} U^\nu$, we obtain for the read response time

$$\begin{aligned} R_r &= D_r + \overline{D} \sum_{\nu=1}^{\infty} U^\nu \\ &= D_r + \overline{D} \frac{U}{1-U} \\ &= D_r + \frac{U_r D_r + U_w D_w}{1-U} \\ &= \frac{D_r + U_w (D_w - D_r)}{1-U}. \end{aligned}$$

An update operation accesses first the message disk. Once, the message disk is read, the Δ-value is available and the process of writing $w-1$ check disks can be started.

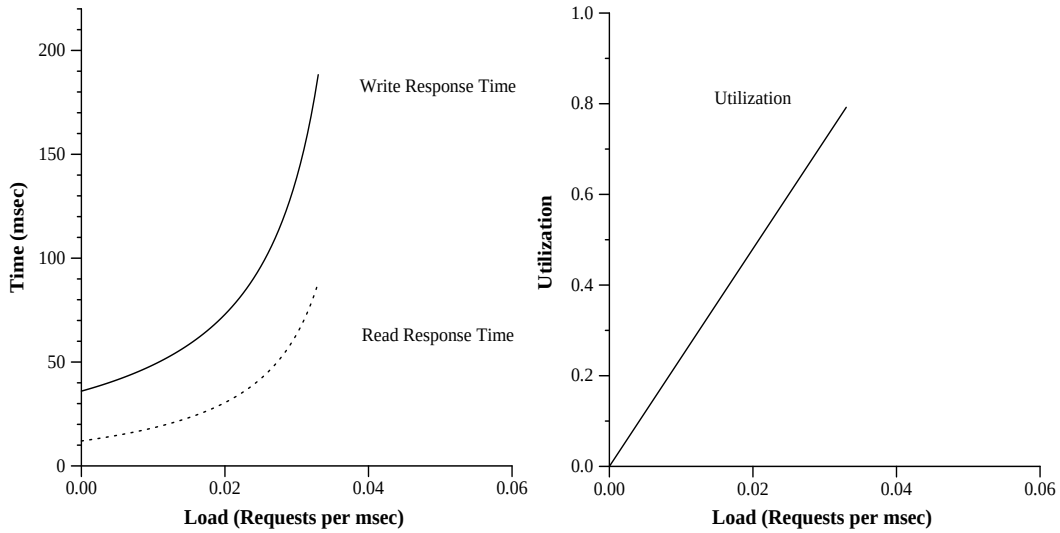


Figure 10.1: Performance with No Write Synchronization

If we assume that response times are approximately exponentially distributed, we can use the harmonic number H_{w-1} as a multiplier to calculate the response time for the last of the $w-1$ check disk accesses. We arrive at

$$R_w = \left(\frac{D_r + U_w(D_w - D_r)}{1-U}\right) + H_{w-1}\left(\frac{D_w + U_r(D_r - D_w)}{1-U}\right)$$

as the response time for a write request. The first addend estimates the response time for the Δ-value to be available and the last one estimates the response time for the check writes. The write at the message disk finishes exactly two latencies after the old information data has been read and is always done before any check disk.

We now consider a level 4 RAID which features fixed message and check disk assignment. Assume that C of the N disks are check and consequentially $N-C$ message disks. The load at a message disk is $\rho_r\Lambda/(N-C)$ reads and $\rho_w\Lambda/(N-C)$ writes. At a check disk, it is $(w-1)\rho_w\Lambda/C$ and consists only of writes. The utilization at a message disk consists of read load $U_{m,r}$ and write load $U_{m,w}$ and is expressed by

$$U_m = (\rho_r D_r + \rho_w D_w)\Lambda/(N-C)$$

and at a check disk

$$U_C = (w-1)\rho_w D_w \Lambda/C.$$

The read response time is

$$R_r = \frac{D_r + U_{m,w}(D_w - D_r)}{1 - U_m}.$$

The write response time is

$$R_w = \frac{D_r + U_{m,r}(D_r - D_w)}{1 - U_m} + H_{w-1}\frac{D_w}{1 - U_c}.$$

We present the response times for a Level 5 RAID without write synchronization in Figure 10. We use a write service time of 24 msec, a read service time of 12 msec and a read to write ratio of 2:1. The load is the number of arriving request per msec. The queuing network remains stable for loads of up to .04 requests per msec and disk, but we give results only for loads less than 0.33, because the corresponding write response time would exceed 200 msec.

Chapter 11

Adjustment for the Presence of Failed Components

Performance worsens considerably after component failure. We describe the effects analytically and give examples. The effects of faults depends strongly on the type of RAID and the existence and kind of spare space. We organize the results in a section for RAIDs using ACATS and one using the classic, static reliability group assignment. The results for less important types (Level 4, Complete address translation) can be derived with ease from our calculations. Within each section we subdivide according to the sparing scheme that is used. Finally, we treat the cases of the "write quorum" $w = 2$ and $w > 2$ and make thus the distinction between an MDS based RAID and two dimensional RAIDs on one side and a traditional one-dimensional RAID on the other side.

We only treat the case of a single disk and a single string failure. The more exotic failure modes are so rare, that they have no influence on the performance and it comes as no surprise that our more resilient RAID types can suffer failure modes that lead to bad performance. In all cases, it is easy to adjust our treatment to these rare modes.

11.1 Strategies for RAIDs with Spare Space

The reconstruction of data originally stored on the lost message disk on the spare disk (or spare space) determines the load and hence performance immediately after the occurrence of a failure.

Symbol	Explanation	Example Value
n	Number of Strings.	10 — 11
m	Number of Reliability Groups.	5
w	Number of Disks accessed during a Write	
N	Numbe of Disks	nm
T	Number of Tracks	2000
ζ	User Request Activated Reconstruction Load.	
κ	Portion Reconstructed or Reconfigurated.	0 — 1
λ_r	Read Load at a Disk.	10 ms
λ_w	Write Load at a Disk.	20 ms
λ_t	Track Read Load at a Disk.	15 ms
λ_{tw}	Track Write Load at a Disk.	25 ms
π	Probability of Accessing a Reconstructed Block	0 — 1
ϕ	Forced Reconstruction Load.	
ρ_r	Read Proportion of Requests	0.667
ρ_w	Write Proportion of Requests	0.333
Λ	RAID Load	

Table 11.1: List of Symbols for Chapter 11

On one end of the spectrum we have "lazy" or rather "opportunistic" methods, which use "read redirect" and "write piggy-backing" ([25].) On the other end we find forced reconstruction which rebuilds data independently of demands for data on the lost disk.

In read redirect, data originally stored on the lost disk is read. It is reconstructed with a read to enough disks in the same reliability group and the result is not only returned in answer to the request but also written to the spare that replaces the failed disk. Write piggy-backing writes data that would have been stored on the failed disk directly to the spare. In order to maintain the consistency of the check information, we need to read all or almost all the disks in the same reliability group. We can use write redirect also if a write requests would have changed check information on the failed disk. In this case, too, we need to access all the message disks in the reliability group.

In forced reconstruction, the RAID controller issues reads to the failed disk that result in the reconstruction of the data on the lost disk, which is then written to the spare. The advantage of using only forced reconstruction is administrative: The reconstructed data blocks are contiguous, whereas read redirect and write piggy-backing reconstructs blocks seemingly randomly. For all reconstruction schemes it is advantageous to use whole tracks instead of blocks. The advantage of using the opportunistic methods consists of the labour

savings involved and the effect of temporal locality, which reconfigures the more frequently used data first on spare space.

Every RAID needs to transform user request addresses to a block of data into disk addresses. In the absence of faults, we only use a simple formula to accomplish the translation. In the presence of faults at least a list of failed or inaccessible disks is required together with their reliability group. The latter can be implemented with a simple formula. The location data increases with the use of sparing and especially with distributed sparing. The mode of data reconfiguration on spares is another drain on the storage needs of the RAID. If forced reconstruction is used and the reconstruction proceeds linear, e.g. from the highest track number to the lowest, only one track or block number needs to be stored. On the other hand, if redirection of reads and piggy-backing of writes is used, reconstructed addresses need to be stored to make use of this procedures. While most of the control data is stored in the RAID non-volatile storage, during reconfiguration of a whole string as an extreme example, some of these data needs to be written to disk and thus increases the RAID load at a point when already the resources are strained. We exclude this kind of load from our considerations, because the trade-offs between hardware (larger RAID controller) and performance have to be assessed in a late stage of RAID design.

Any analysis of a scheme that uses write piggy-backing or read redirect needs to be concerned about the temporal locality of accesses. We assume a rule $Z(n)$ that gives the total number of blocks accessed after n accesses total. This rule can reflect *Zipf's distribution* or the 80/20 rule which claims that 20% of the blocks are accessed 80% of the time.

11.2 RAIDs with Almost Complete Address Translation

The RAIDs that use the almost complete address translation scheme (ACATS) can be Level 5 RAIDs, MDS-Code based RAIDs or our two-dimensional schemes. They can use free-standing hot spare disks, distributed sparing of one or two disks in two flavors, naive and safe, and distributed sparing of a whole spare string. Of the many failure modes a RAID can possibly suffer, we limit ourselves to the discussion of the effect of one failure of a disk or a string at a time.

We assume that n disks form a reliability group and that normally w of them need to be written, so that we have $w-1$ checks. The number of reliability groups is m. All n disks in a reliability group are located on the n different strings.

11.2.1 No Spares

When no spares are used, the load adjustment consists of increased read loads due to reads to the failed disk(s), decreased write loads when check information for the written block was kept on the failed disk(s) and increased write loads when the information data was kept on the failed disk.

We concentrate on performance after a single disk or a single string failure has occurred. As only the disks outside the string on which the failed disk is located experience changed load, we derive the loads and then the utilities for them only. We count single writes which do not read the overwritten data first as reads because they are served with the same service time D_r.

No Spares: Write quorum $w = 2$:

Load increases after failure stem from three kind of requests: reads to the failed disk, writes to the failed disk, which serves as the message disk, and writes to the failed disk, when it serves as the check disk. We service the first kind (reads to the failed disk) by reading the $n-1$ other disks in the reliability group and recovering the information that then is relayed to the requestor. If a write addressed to the failed disk in capacity of a message disk arrives, we need to update the check information. We have to read $n-2$ disks and then perform a simple write at the check disk. Because we need the results of the reads first, the write to the check disk can be modeled as an additional disk access. It has the same service time as a read and is counted as a read. If we service a write that uses the failed disk as a check disk, we only need to do a single write at the addressed disk. We count this as a read. The frequency of these requests are respectively $\rho_r\Lambda/nm$, $\rho_w\Lambda/nm$ and $\rho_w\Lambda/nm$.

Because writes involving the failed disk are treated differently, we need to calculate this load at the disks outside the string which contained the failed disk. There are $\rho_w\Lambda/nm$ write requests directed to the failed disk in capacity of message disk and the same amount in capacity of check disk. These correspond to the same number of requests to the $(n-1)m$

disks outside the string with the failed disk.

Summing up, we obtain for the new read and write load at the disks outside the string which contained the failed disk:

$$\begin{aligned}
\lambda_r^{df} &= \frac{\rho_r\Lambda}{nm} + \frac{n-1}{(n-1)m}\frac{\rho_r\Lambda}{nm} + \frac{n-1}{(n-1)m}\frac{\rho_w\Lambda}{nm} + \frac{1}{(n-1)m}\frac{\rho_w\Lambda}{nm} \\
&= \left(1+\frac{1}{m}\right)\lambda_r^{old} + \frac{n}{(n-1)m}\lambda_w^{old} \\
\lambda_w^{df} &= \frac{2\rho_w\Lambda}{nm} - \frac{2\rho_w\Lambda}{nm(n-1)m} \\
&= (1-\frac{1}{(n-1)m})\lambda_w^{old}
\end{aligned}$$

In the first equation, we have first the normal read load, the additional read load generated by read requests to the failed disk, the accesses caused by writes to the failed disk as a message disk and finally the writes that used the failed disk as a check disk. The diminished write load can actually better performance, if the strong write synchronization is used. The load increase is approximately $1/m$ the original load.

No Spares: Write Quorum $w > 2$:

In this section we investigate the performance for the MDS and the two-dimensional RAID. We need to consider again the three kinds of requests that will change the load. A read to the failed disk is handled by accessing $n-w+1$ disks. A write to the failed disk as message disk still needs to access $n-w$ disks and then write to $w-1$ disks simultaneously. These writes are synchronized, but we only need to overwrite the old check data with the new values. The synchronization method is crucial here and we derive the loads only for the no-write-synchronization scheme. Then we can treat the $w-1$ writes as independent disk accesses, which count as reads. Because it is not likely that there is an interfering request between the read and the write, this approximation is slightly pessimistic. An exact solution can easily be obtained, but suffers from an undue multiplication of request classes. A write to the failed disk as check disk needs only to perform a usual write with only $w-1$ participating disks. As we have already restricted ourselves to the no-write-synchronization scheme, we account for these among the write requests.

We have $\rho_r\Lambda/nm$ read requests to the failed disk, $\rho_w\Lambda/nm$ write request to the failed disk in its capacity as a message disk and $(w-1)\rho_w\Lambda/nm$ write requests to it as a

check disk. The original load of write requests involving the failed disk as either message or check disk is $(w-1)w\rho_w\Lambda/nm(n-1)m$.

We have

$$\begin{aligned}
\lambda_r^{df} &= \frac{\rho_r\Lambda}{nm} + \frac{n-w+1}{(n-1)m}\frac{\rho_r\Lambda}{nm} + \frac{n-w+w-1}{(n-1)m}\frac{\rho_w\Lambda}{nm} \\
&= \left(1 + \frac{n-w+1}{(n-1)m}\right)\lambda_r^{old} + \frac{1}{mw}\lambda_w^{old} \\
\lambda_w^{df} &= \frac{w\rho_w\Lambda}{nm} - \frac{w-1}{(n-1)m}\frac{w\rho_w\Lambda}{nm} + \frac{w-1}{(n-1)m}\frac{(w-1)\rho_w\Lambda}{nm} \\
&= \left(1 - \frac{w-1}{w(n-1)m}\right)\lambda_w^{old}
\end{aligned}$$

In the first equation, the first addend represents the normal read load, the second reads to the failed disk and the third writes to the failed disk in the capacity of message disk. The second equation's summands represent the normal workload, then writes that involve the failed disk and the writes to blocks of which one check disk was the failed disk.

No Spares: Adjustment for String Failure

We obtain our result by analogue assumptions and reasoning. Now, the additional work is m times as large as before. We have for $w = 2$:

$$\begin{aligned}
\lambda_r^{sf} &= \frac{\rho_r\Lambda}{nm} + \frac{(n-1)m}{(n-1)m}\frac{\rho_r\Lambda}{nm} + \frac{(n-1)m}{(n-1)m}\frac{\rho_w\Lambda}{nm} + \frac{m}{(n-1)m}\frac{\rho_w\Lambda}{nm} \\
&= 2\lambda_r^{old} + \frac{n}{(n-1)}\lambda_w^{old} \\
\lambda_w^{sf} &= \frac{2\rho_w\Lambda}{nm} - \frac{2m\rho_w\Lambda}{nm(n-1)m} \\
&= (1 - \frac{1}{(n-1)})\lambda_w^{old}
\end{aligned}$$

and for $w > 2$:

$$\begin{aligned}
\lambda_r^{sf} &= \frac{\rho_r\Lambda}{nm} + \frac{(n-w+1)m}{(n-1)m}\frac{\rho_r\Lambda}{nm} + \frac{(n-w+w-1)m}{(n-1)m}\frac{\rho_w\Lambda}{nm} \\
&= \left(1 + \frac{n-w+1}{(n-1)}\right)\lambda_r^{old} + \frac{1}{w}\lambda_w^{old} \\
\lambda_w^{sf} &= \frac{w\rho_w\Lambda}{nm} - \frac{(w-1)m}{(n-1)m}\frac{w\rho_w\Lambda}{nm} + \frac{(w-1)m}{(n-1)m}\frac{(w-1)\rho_w\Lambda}{nm} \\
&= \left(1 - \frac{w-1}{w(n-1)}\right)\lambda_w^{old}
\end{aligned}$$

No Spares: An Example

We illustrate our results in an example that shows the typical utilization adjustments. We consider a small RAID with 5 reliability groups distributed over 10 strings. Each reliability group has only one check disk. There are 45 message disks and 5 check disks in the RAID for a total of 50 disks. Alternatively, we consider an MDS-based RAID with 11 strings and 5 reliability groups. The two RAIDs have the same data storage capacity. We base our service time estimates on a latency of 5 ms and an average seek time little less than 5 ms. This gives

$$D_r = 10 \text{ ms} \qquad D_w = 20 \text{ ms}$$

We assume a read ratio of 66.67% in the RAID load mix. If the RAID load is Λ, measured in requests per millisecond, the utilization of the disks is 0.4Λ for the one-dimensional RAID and $.485\Lambda$ for the MDS-RAID. After a disk failed, the utilization goes up to 0.450Λ for the one-dimensional RAID and to 0.518Λ for the MDS-based RAID. For a string failure, the values are 0.652Λ and 0.691Λ, respectively. These numerical results depend on the dimensions of the RAID.

11.2.2 Reconfiguration on Check

If no spare space can be provided, we can increase performance by reassigning the check tracks to contain message data originally stored on the lost disk. The draw-back to this scheme is the necessity to restore the original RAID lay-out in the repair procedure. (If we do not do this, then the replacement disk will keep all the check data and we end up with a RAID 4 organization's bad performance.) We call the scheme presented here "Reconfiguration on Check" in contrast to "Reconstruction on Spare" which we discuss below. In Section 11.2.3 we will discuss possible reconfiguration strategies at length. Here, we just analyze a particular scheme that achieves excellent reconfiguration times. In all cases, the RAID controller needs to keep track of the reconfiguration progress. First, we treat the failure of one disk.

A bitmap contains the information whether the check track in a reliability group carries reconfigured message data from the failed disk or check data. During any operation that accesses message data on the failed disk, we expand the data reconstruction procedure by overwriting the check track with the message track from the failed disk. By accessing

whole tracks instead of blocks we speed up the reconfiguration procedure considerably. Another advantage is that spatial locality increases the hit rate of finding the data already on the check disk. In addition, we use "forced reconfiguration" in which we reconstruct a data track of the failed disk independently of user requests.

We denote by κ the proportion of the check tracks that now carry the message data from the failed disk. If there are T tracks, then there will be only $T \cdot (n-1)/n$ check tracks, because one n^{th} of the check tracks were located on the failed disk. We can (very roughly) estimate the hit ratio to be equal to κ. Actual behavior differs somewhat for small κ the hit ratio is much higher, but with larger κ it will start to lag behind. The reconfiguration times are minutes and a typical user might not even be logged on, to say nothing of accessing all the data files in such a short time. We hence use κ as the hit ratio as a convenient fiction, that does not falsify our conclusions, as long as we do not suffer the illusion, that forced reconfiguration will not eventually become necessary if we want to replace all check tracks by message tracks.

We call ϕ the forced reconfiguration load and ζ the user request activated reconfiguration load.

Reconfiguration on Check: Write quorum $w = 2$:

The user request activated reconfiguration load is

$$\zeta = (1-\kappa) \cdot (\lambda_r^{old} + \frac{1}{2} \cdot \lambda_w^{old})$$

because only reads and writes to the failed disk in its capacity of message disk will generate a reconfiguration of data on check.

The only disks that have a track in the same reliability group as a track on the failed disk are the disks outside the string on which the failed disk was located. The utilization at these disks will be worse and we concentrate our analysis on them.

Because we are reconfiguring all reliability groups that have lost a track on the failed disk, writes in these groups are now a simple overwriting of the old message data. We treat these reads in our analysis as reads, because they proceed with the same service time. On average, a disk will serve the afflicted reliability group $1/m$ times. Hence, the normal

service loads are

$$\begin{aligned} \text{Reads:} \quad & \frac{m-1}{m} \cdot \frac{\rho_r \Lambda}{nm} + \frac{1}{m} \cdot \frac{\Lambda}{nm} \\ \text{Writes:} \quad & \frac{m-1}{m} \cdot \frac{2\rho_w \Lambda}{nm} \end{aligned}$$

We need to add to this the load originally directed to the failed disk, which will be spread over the $(n-1)m$ disks not in the same string as the failed disk. We obtain:

$$\begin{aligned} \text{Reads:} \quad & \frac{m-1}{m} \cdot \frac{\rho_r \Lambda}{nm} + \frac{1}{m} \cdot \frac{\Lambda}{nm} + \frac{\kappa}{(n-1)m} \frac{\Lambda}{nm} \\ \text{Writes:} \quad & \frac{m-1}{m} \cdot \frac{2\rho_w \Lambda}{nm} \end{aligned}$$

The final adjustment introduces two new classes of requests: The first is the track read, whose service time D_t is the arithmetic average between a read and a write request, because it contains two disk latencies as opposed to one for the read and three for the write operation. we denote the load by λ_t. The second class are the track writes with service time D_{tw} which contains 4 latencies. During a reconfiguration operation, we first read all the message disks available and then read and rewrite the check track. We then obtain for the final count:

$$\begin{aligned} \lambda_r &= \lambda_r^{old} + \frac{(n-1+\kappa)\Lambda}{(n-1)mnm} \\ \lambda_w &= \frac{m-1}{m} \lambda_w^{old} \\ \lambda_t &= \frac{(n-2)(\phi+\zeta)}{(n-1)m} \\ \lambda_{tw} &= \frac{\phi+\zeta}{(n-1)m} \\ \zeta &= (1-\kappa)\frac{\Lambda}{nm} \end{aligned}$$

We will see a slight utilization increase right after the failure and then see the utilization reaching a lower level (as the hit probability $\pi \to 1$) because $1/m$ of the writes no longer require updates of check data. This saving is slightly offset by the additional load resulting from the concentration of all message data on the surviving disks in the reliability group.

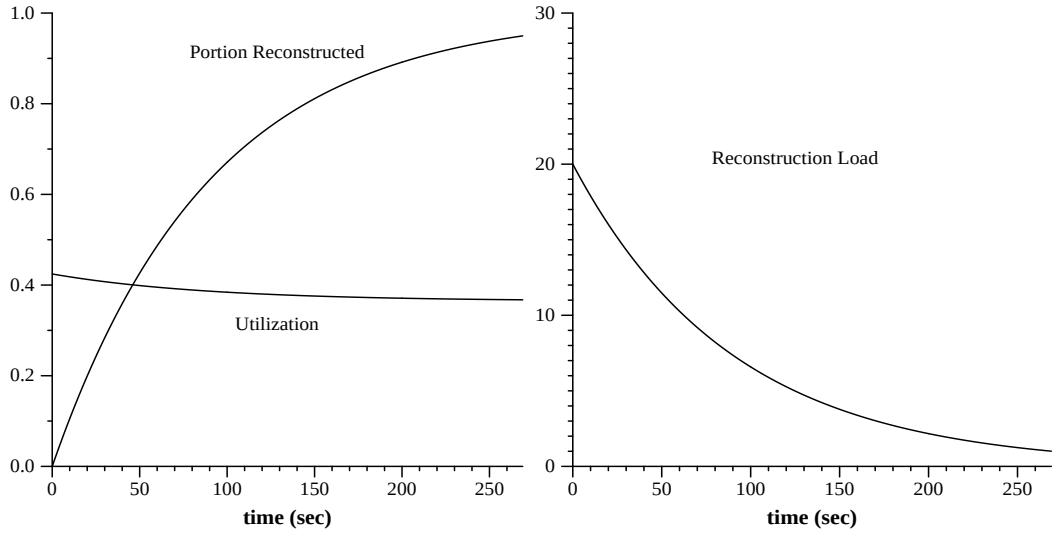


Figure 11.1: Reconfiguration effects on the Utilization of the Disks outside the String with the Failed Disk for Reconfiguration on Check at a one-dimensional RAID. We use opportunistic reconstruction only.

Reconfiguration on Check: Example (continued)

In our example, we will see the utilization increase to .42444Λ or by 6.11% of the baseline utilization of .4Λ immediately after the disk failure, when $\kappa = 0$. Then slowly the utilization will fall to 0.36444Λ or by 8.88% of the baseline utilization. The latter number consists of 10.00% savings due to writes that do not need to update check information as well and an additional utilization of 1.11% in order to replace the failed disk. The utilization of the disks on the same string stays constant at the baseline level of 0.400Λ.

We present the utilization during the reconfiguration process at the disks outside the string with the failed disk in Figures 11.1 and 11.2. (Because the actual behavior depends on locality of data accesses, the times in Figure 11.1 and in all Figures presenting are not a good predictor of actual behavior, but the utilization behavior over time is an excellent predictor.

Reconfiguration on Check: Write Quorum $w > 2$:

Most RAID schemes built are one-dimensional RAIDs with one string of spares at the most. An MDS RAID has already committed the same resources and further addition of spare

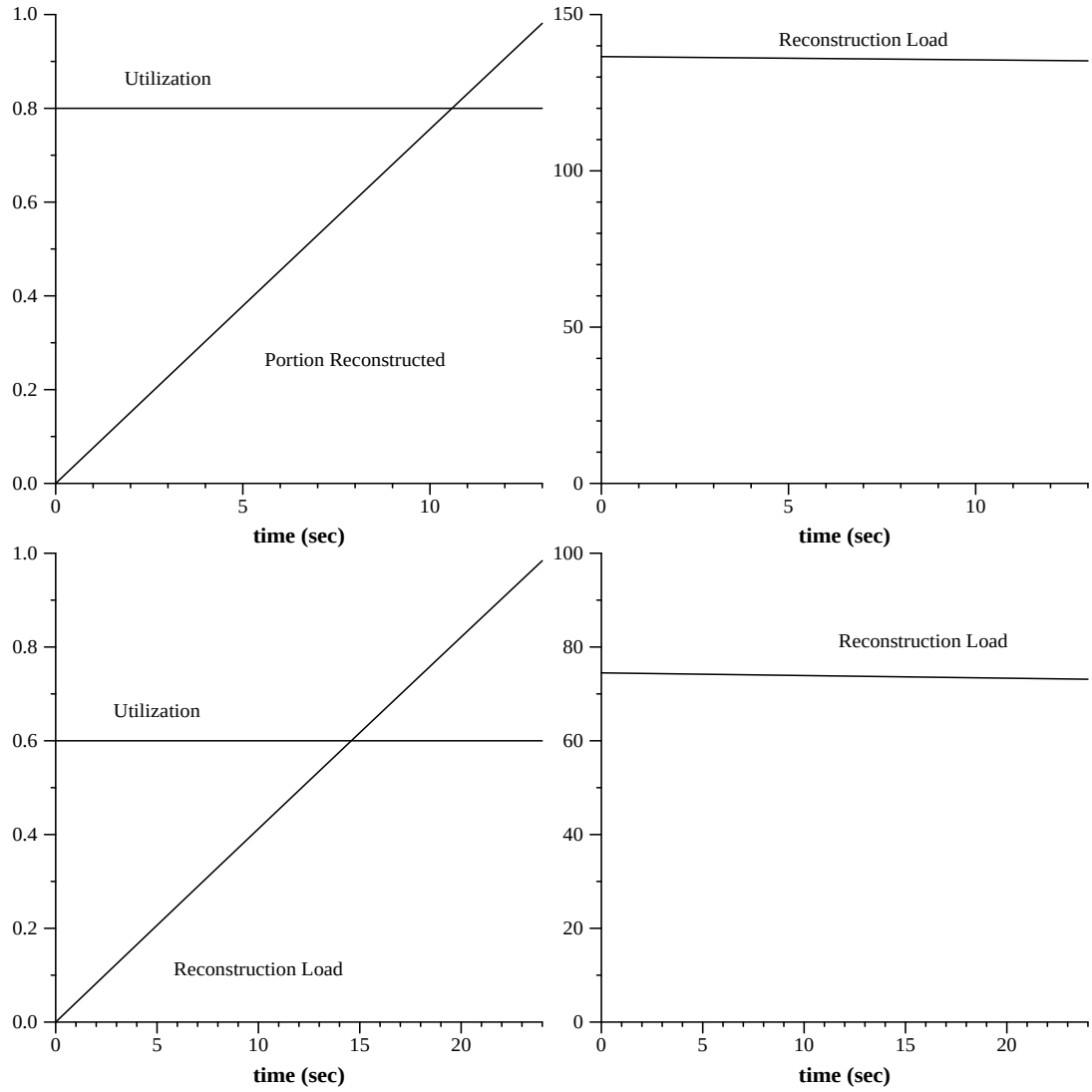


Figure 11.2: Reconfiguration effects on the Utilization of the Disks outside the String with the Failed Disk for Reconfiguration on Check at a one-dimensional RAID. The utilization of the disks outside the string with the failed disk is kept at 80% (top) and 60%(bottom).

disks or strings is not necessary advantageous. Reconfiguration of an MDS based RAID is then a very attractive way of using the (by comparison) generous redundancy that the MDS RAID provides. This appraisal is reinforced by the design philosophy of the MDS RAID which tries to avoid the effects of multiple, simultaneous component failures, which becomes impossible after failure.

A reliability group in an MDS RAID contains at least two virtual check disks. Only one of them is being overwritten with reconstructed information data. If some track on the failed disk contains check data from the other virtual disk, we do not transfer these, but then the RAID controller is slightly more complex.

To analyze the performance in complete generality, we would have to introduce a class of writes, which involves only $w-1$ in lieu of w disks. If we do not use strong write synchronization, (which we have shown to have a very adverse effect on performance,) we do not need this distinction. Therefore, we assume that strong synchronization is not used.

The read load consists of the old read load plus π times the read load at the failed disk distributed over $(n-1)m$ disks. The write load changes because in the afflicted reliability groups the writes now include only $w-1$ disks and because the lighter writes to the failed disk are not part of the load at the disks outside the string with the failed disk. If a read tries to access data on the failed disk, this request leads to a data reconstruction ($n-w$ track reads and 1 track read + write.) The same happens, if we try to write message data to the failed disk, before the track has been reconfigured on check space.

We obtain for the loads at the disks outside the string with the failed disk:

$$\begin{aligned}
\lambda_r &= \frac{\rho_r\Lambda}{nm} + \frac{\pi}{(n-1)m}\frac{\rho_r\Lambda}{nm} \\
&= (1+\frac{\pi}{(n-1)m})\lambda_r^{old} \\
\lambda_w &= \frac{m-1}{m}\frac{w\rho_w\Lambda}{nm} + \frac{1}{m}\frac{(w-1)\rho_w\Lambda}{nm} + \frac{\pi}{(n-1)m}\frac{(w-1)\rho_w\Lambda}{nm} \\
\lambda_t &= \frac{(n-w)}{(n-1)m}(\zeta+\phi) \\
\lambda_{tw} &= \frac{1}{(n-1)m}(\zeta+\phi) \\
\zeta &= (1-\pi)\left(\frac{\rho_w\Lambda}{nm} + \frac{\rho_r\Lambda}{nm}\right)
\end{aligned}$$

At the beginning of the reconfiguration process, the utilization will exceed the

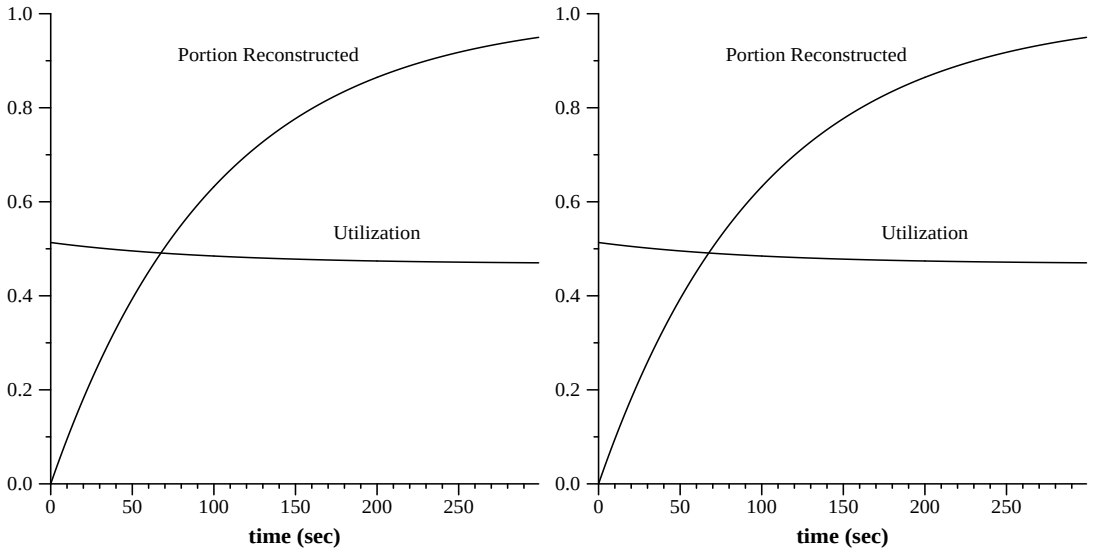


Figure 11.3: Reconfiguration effects on the Utilization of the Disks outside the String with the Failed Disk for Reconfiguration on Check at a MDS based RAID. We use only opportunistic reconstruction.

utilization if we do not reconfigure, but then the utilization will decrease to a level lower than baseline. Only the disks outside the string with the failed disk profit from this savings, so that the performance gain is partially not realized.

Reconfiguration on Check: Example (continued)

We continue our example for the MDS RAID. If we use opportunistic reconstruction only, the initial utilization increases from 0.485Λ to 0.513Λ as compared to 0.518Λ without reconfiguration and finally settles at 0.470Λ. The latter figure reflects the lower write load, because one reliability group performs updates with writes to one disk less. This savings exceeds the load increase resulting from the distribution of the load of the failed disks over all strings but the one with the failed disk. We give numerical examples in Figures 11.3 to 11.4.

Reconfiguration on Check: String Failure

We now adjust our analysis to treatment of reconfiguration in the string failure case. The modifications to the original load now have to be multiplied essentially by a factor of m.

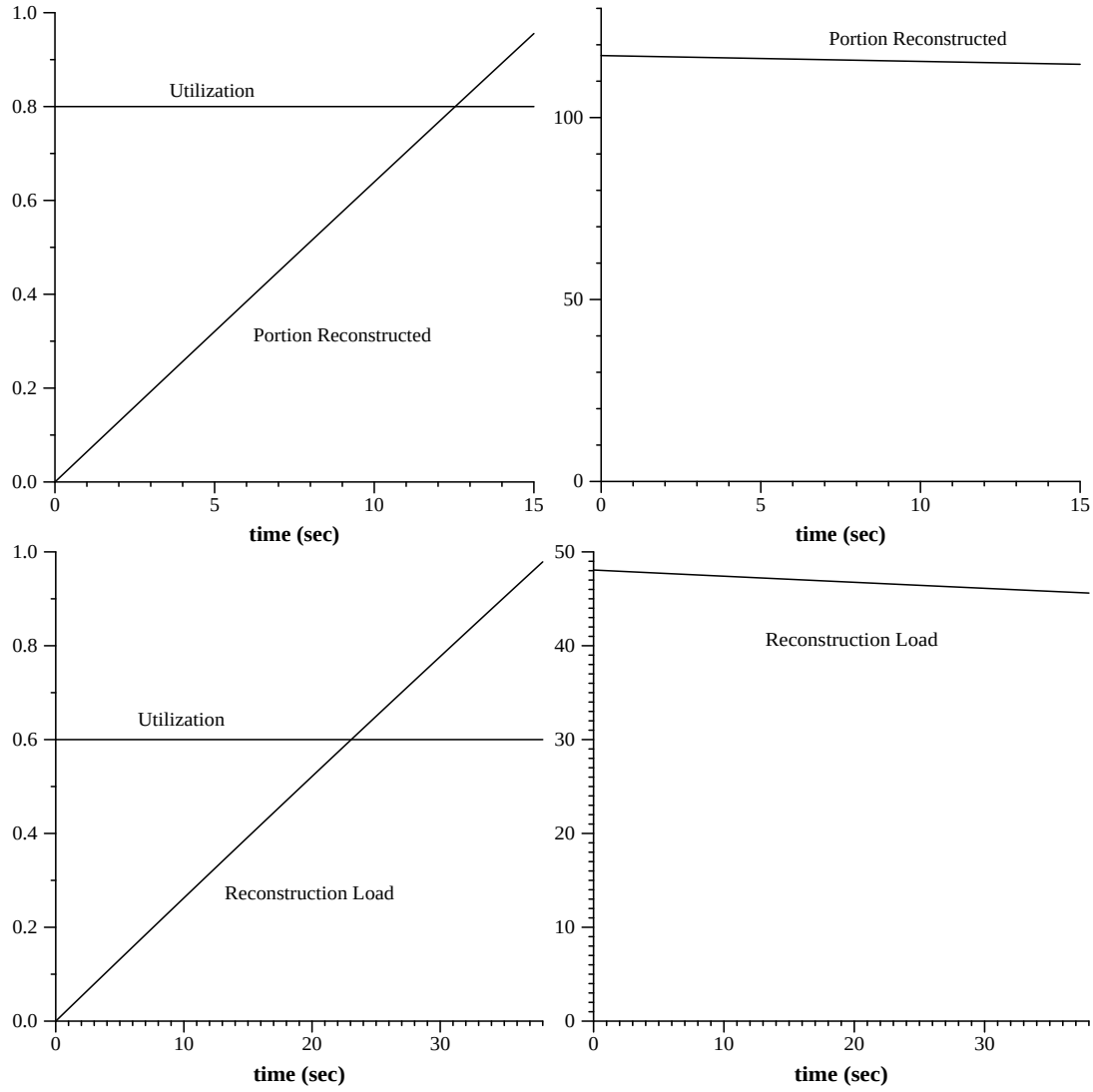


Figure 11.4: Reconfiguration effects on the Utilization of the Disks outside the String with the Failed Disk for Reconfiguration on Check at a MDS based RAID. The reconfiguration proceeds at constant disk utilization of 80% (top) and 60% (bottom) in all strings but the one with the failed disk.

Reconfiguration on Check: Write Quorum $w = 2$

With the same notation as in the disk failure case treated above we have

$$\begin{aligned}
\lambda_r^{sf} &= \frac{\Lambda}{nm} + \frac{\pi}{n-1}\frac{\Lambda}{nm} \\
\lambda_w^{sf} &= 0 \\
\lambda_t^{sf} &= \frac{(n-2)(\phi+\zeta)}{(n-1)} \\
\lambda_{tw}^{sf} &= \frac{\phi+\zeta}{(n-1)} \\
\zeta &= (1-\pi)\frac{\Lambda}{nm}
\end{aligned}$$

At the beginning of the reconfiguration process, the utilization increases, but not as much as without reconfiguration. We are reaping here the benefit of avoiding check disks update. After reconfiguration we are left with a disk array without redundancy. The utilization is a rather reduced to

$$u^{rcf} = \frac{\Lambda}{(n-1)m} D_r.$$

This makes reconfiguration a most attractive scheme. At the end of the repair, the RAID needs to reconstruct the lost check data, which can be done at a relatively low disk utilization and hence without serious performance loss.

Reconfiguration on Check: Example (continued)

In our example, the utilization at the beginning of the reconfiguration process is 0.522Λ. After the reconfiguration the utilization is reduced to 0.222Λ. We give examples of the reconfiguration process in Figures 11.5 and 11.6.

Reconfiguration on Check: Write Quorum $w > 2$

After the string failure, all writes will proceed with one less write to disk. With this in mind, we can give the write load exactly even for a use of the strong write synchronization scheme. The loads are

$$\begin{aligned}
\lambda_r^{sf} &= \lambda_r^{old} + \frac{\pi}{(n-1)}\lambda_r^{old} \\
\lambda_w^{sf} &= \frac{(w-1)\Lambda}{nm} + \frac{\pi}{(n-1)}\frac{(w-1)\Lambda}{nm} \\
\lambda_t^{sf} &= \frac{n-2}{n-1}(\zeta+\phi)
\end{aligned}$$

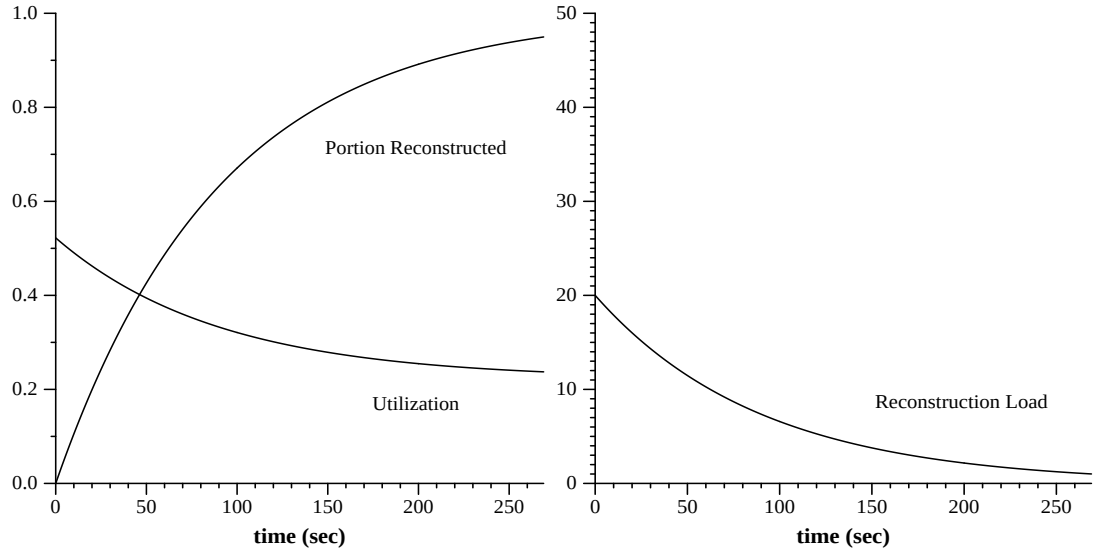


Figure 11.5: Reconfiguration Effects on the Utilization of the Disks after a String Failure in the One-Dimensional RAID Organization. We use only opportunistic reconstruction.

$$\lambda_{tw}^{sf} = \frac{1}{n-1}(\zeta + \phi)$$

The utilization at the beginning of the reconfiguration process is slightly higher than without reconfiguration, but at the end stabilizes at the utilization of a RAID with one less check disk, which is always lower. Again, reconfiguration turns out to be a very attractive way to handle component failure. The only disadvantage is a slightly more complicated and hence insignificantly longer repair time. We can use almost arbitrarily small utilization increases to restore the original RAID configuration at the end of a repair.

Reconfiguration on Check: Example (continued)

Returning to our example, we present the result of our simulations for the MDS RAID in Figures 11.7 and 11.8. As before, they show reconfiguration to be very attractive.

11.2.3 Use of a Spare Disk

In this section we assume the use of a spare disk. We calculate the load when a proportion κ of the failed disk is already reconstructed. We first treat the case of a failed

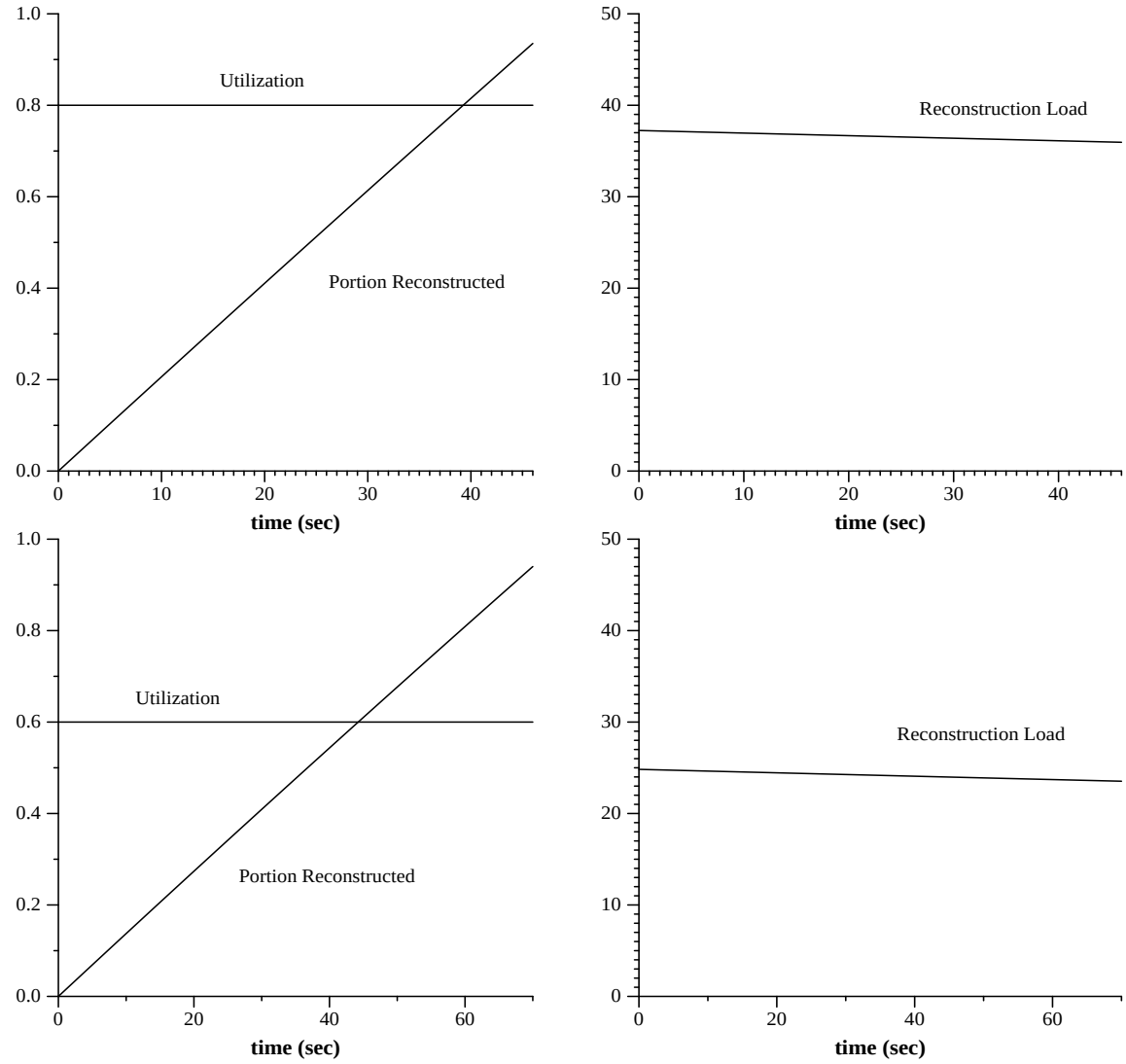


Figure 11.6: Reconfiguration Effects on the Utilization of the Disks after a String Failure in the One-Dimensional RAID Organization. We use constant disk utilization at 80% (top) and 60% (bottom).)

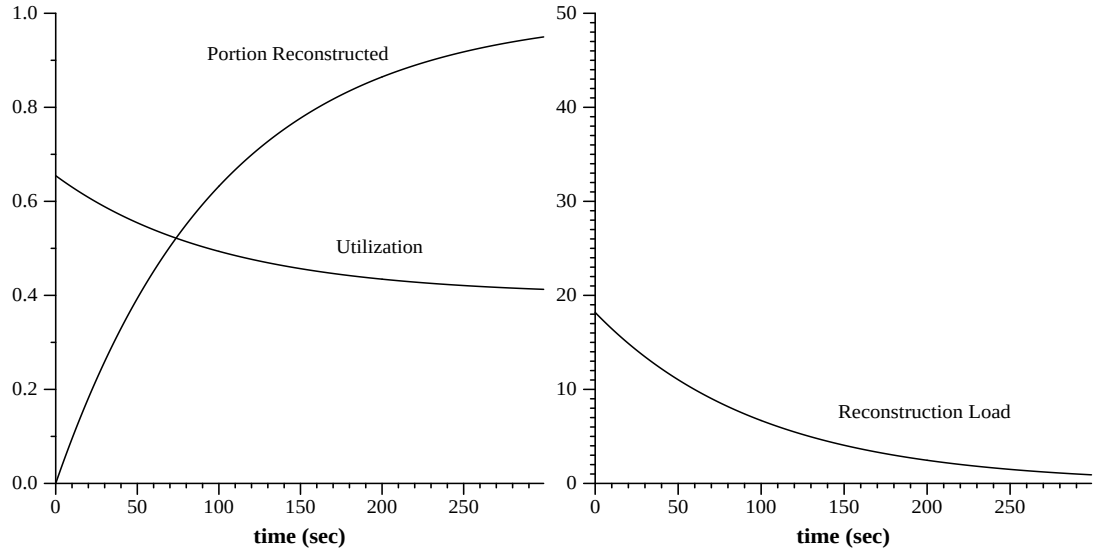


Figure 11.7: Reconfiguration effects on the Utilization of the Disks outside the String with the Failed Disk for Reconfiguration on Check in a MDS-based RAID. We use only opportunistic reconstruction. (The reconstruction load is given in requests per sec/100.)

disk. (A similar analysis is given in [25]. In contrast to *loc. cit.*, we reconstruct full tracks and we also assume that utilization is relatively low and should never reach 1.)

If we use forced reconstruction at a rate of ζ requests per time unit, an additional read load of $\lambda_{rt} = \zeta/(n-1)m$ will apply. This load will be serviced with a different service time because we read a whole track. This is almost the service time of an unsynchronized write.

We now include read redirect and write piggy-backing in our performance evaluation. To decrease the reconfiguration time S of the stand-by disk, we assume that the whole track is reconfigured with an attempted access to a block on the failed disk.

We had decreed the existence of a rule $Z(N)$ that gives the number of tracks accessed in N accesses. Given the load Λ at the RAID, we can determine the load ζ of opportunistic reconstruction.

The failed disk stored many more message than check data. The check data and the message data are, however, rewritten equally often. Opportunistic reconstruction of check data protects the last written and presumably most important data and is therefore

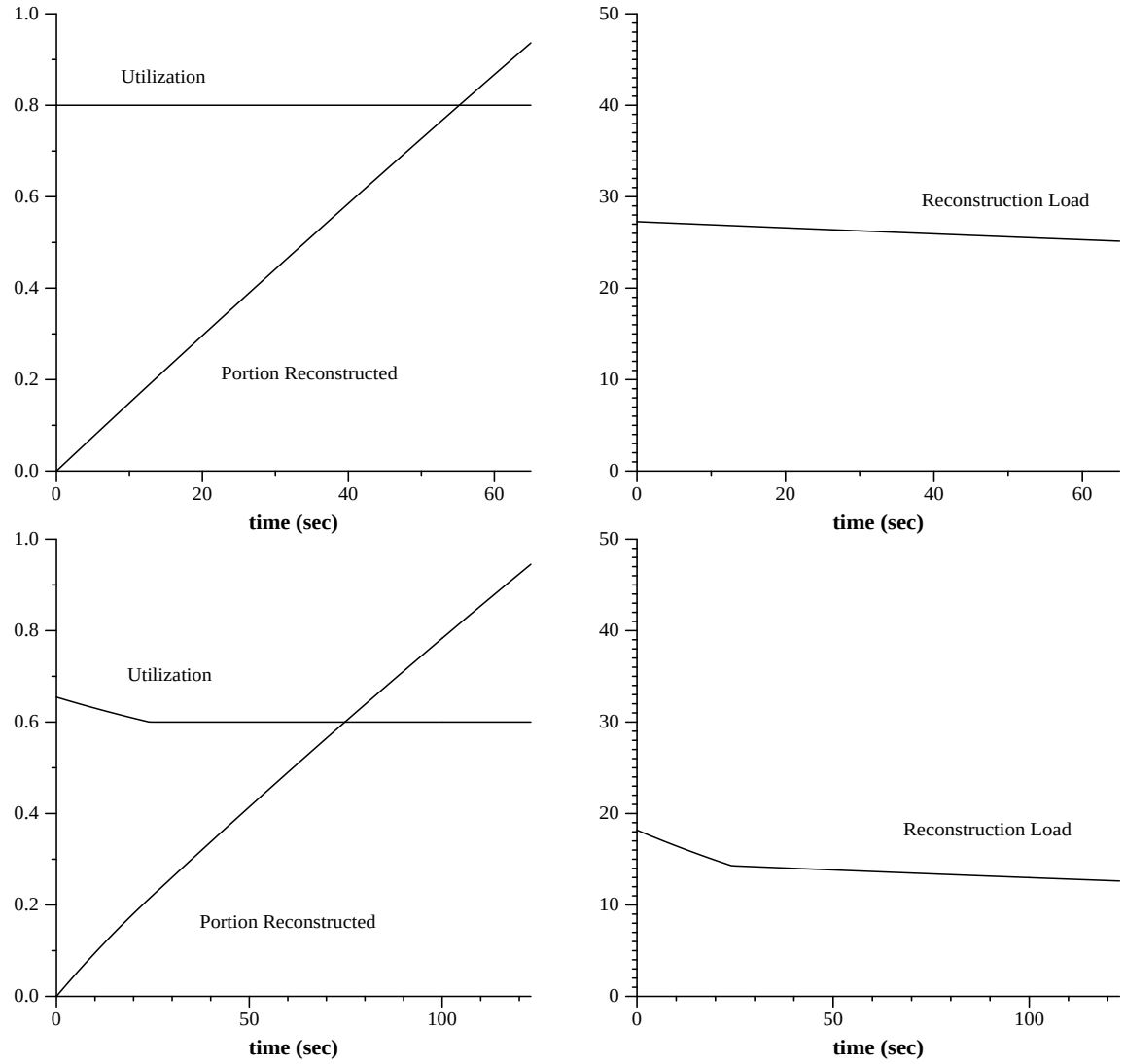


Figure 11.8: Reconfiguration effects on the Utilization of the Disks outside the String with the Failed Disk for Reconfiguration on Check in an MDS RAID. We fix disk utilization at 80% (top) and at 60%.

advantageous. The probability $\pi(t)$ that a track accessed on the failed disk is already reconstructed on the spare, is independent of the character of the write (i.e. whether the failed disk is message or character disk) and presumably independent of the character of the request (read or write.) A pessimistic model assumes that no temporal locality is seen and obtains

$$\pi(t) = \kappa(t).$$

We have

$$\kappa(t) = f(t)/T + \int_0^t (1 - f(\tau)/T)\zeta(\tau)/T d\tau.$$

We could calculate the reconfiguration time S from this integral equation.

Our analysis applies strictly only to RAIDs that do not use the strong write synchronization scheme, but it can be easily transferred to this scheme. We choose a strategy that reconstructs whole tracks on the spare at once. This strategy wins out over individual block construction, unless the utilization of the RAID is close to one and only opportunistic reconstruction can be performed. We introduce a new kind of request for which a whole track is being read. We indicate this in our formulae by the suffix t, for example, the load of this request at an individual disk is called λ_t.

Use of a Spare Disk: Write Quorum $w = 2$

We initially consider a write quorum of two. To handle a read to the failed disk, we just read the track at the $n-1$ disks in the same reliability group, store the data temporarily until all are available and then reply the user request and write the whole track to the spare. If a write requests arrives that was addressed to the failed disk as a check disk, we again read $n-1$ tracks in the same reliability group, and reconstruct the check data on the spare disk. In this procedure, we replace the overwritten block with the new data. We execute a write directed to a block originally stored on the failed disk by first reading the track at all message disks in the reliability group. With the written block we can now calculate the corresponding check block. We now access the check track, reading the information at all but the new check block which is written instead. With the information gathered, we can now write all the blocks of the track on the spare. Consequentially, all operations directed to the failed disk cause $n-1$ track reads, potentially interspersed with a single block write.

The disks outside the string with the failed disk will experience a load shift from

write operations to reading tracks due to writes involving the failed disk. The loss of write load is $(1-\pi(t))\cdot\lambda_w^{old}\cdot 1/(n-1)m$ where π is the probability that a write can be satisfied by the spare. The exact value of π depends on the effects of temporal locality, but is probably much smaller than $\kappa(t)$, the portion of the spare that has already been reconstructed. We can obtain an approximation of the disk load by assuming that a write to the failed disk causes $n-2$ track reads and does not change the write load. The proportion of these requests is $2\rho_w/(\rho_r+2\rho_w)$.

We obtain for the loads:

$$\begin{aligned}
\lambda_r^{df} &= \lambda_r^{old} \\
\lambda_w^{df} &= \lambda_w^{old} - \frac{1-\pi(t)}{(n-1)m}\lambda_w^{old} \\
\lambda_t^{df} &= \frac{\phi+\zeta}{m} \\
\zeta &= (1-\pi(t))\lambda^{old}
\end{aligned}$$

and for the approximate loads (denoted with a superfix a):

$$\begin{aligned}
\lambda_r^{adf} &= \lambda_r^{old} \\
\lambda_w^{adf} &= \lambda_w^{old} \\
\lambda_t^{adf} &= \frac{2\rho_w(n-2)+\rho_r(n-1)}{n-1}\cdot\frac{\phi+\zeta}{(n-1)m}
\end{aligned}$$

The load at the spare consists of the reconstruction load and write and read redirects:

$$\begin{aligned}
\lambda_r &= \pi(t)\lambda_r^{old} \\
\lambda_w &= \pi(t)\lambda_w^{old} \\
\lambda_t &= \phi+\zeta \\
\zeta &= (1-\pi(t))\lambda^{old}
\end{aligned}$$

The loads are multiplied with the service times and the sums added up to obtain the utilizations. If we do not use a scheme where the service times depend on the utilization, this becomes a good measure for the performance of the RAID.

The increase of utilization at a disk outside the string with the failed disk is approximately the reconstruction load divided by the number of reliability groups times the average service time, i.e. $(\phi + \zeta)D_t/m$, because the average service time for a typical load with a read to write ratio of 2:1 is exactly the arithmetic average, which turns out to be the service time for a track read. (The typical write has 3 latencies, the typical read 1 and the track read 2.) The opportunistic load is initially the load the failed disk would experience, i.e. λ. We can estimate the utilization before the failure at $u^{old} = \lambda D_t$ and after the failure at least at $u^{df} = \lambda D_t + \lambda D_t/m$, that is, an increase of $1/m$. As more tracks of the failed disk are reconstructed on the spare, the utilization slowly goes back to normal. Utilizations in a RAID rarely exceed 0.3 and this poses no problem. The opportunistic load at the spare is the same as before, but of different character. For a read to write ratio of 2:1 the utilization is not changed. If we introduce forced reconstruction into the picture and for example keep the reconstruction effort constant, the utilization at the disks outside the string with the failed disk will stay at the initial increased level. As the spare fills up with more and more reconstructed tracks, the utilization steadily increases and reaches twice the former level. We can use a different scheme and set the reconstruction workload at a level that keeps the utilization at the spare constant. Then the utilization at the other affected disks increases by approximately $1/m$ the utilization of the spare disk, but slowly decreases while the spare becomes fuller and the reconstruction load is adjusted to allow the spare to serve normal workload requests. The benefits of this procedure do not depend on the exact read to write load ratio. If the temporal locality is high, the spare will receive a higher proportion of requests to the reconstructed tracks and the reconstruction time will be prolonged. If the temporal locality is low or is negative so that recently used tracks are less likely to be accessed, the reconstruction load is higher and the reconstruction time is shortened. (A very similar scheme is presented in [25].)

We calculate the reconstruction time under this policy for a temporal locality of zero, when every track is accessed with the same likelihood independent of past behavior. For our specific read to write requests ratio we derive

$$u_c = \kappa u_o + \psi D_t$$

as an expression for the constant target utilization u_c of the spare. κ stands for the portion of reconstructed tracks on the spare, u_o denotes the old utilization, ψ is the total reconstruction

load ($\psi = \phi + \zeta$) and D_t was the symbol for the track read service time. As

$$\kappa = \frac{1}{T}\int_0^t \psi(\tau)d\tau$$

we obtain the integral equation:

$$u_c = \frac{u_o}{T} \cdot \int_0^t \psi(\tau)d\tau + D_t\psi$$

Differentiation yields

$$0 = \frac{u_o}{T}\psi + D_t\psi'$$

which is solved by the function family

$$\psi(t) = C\exp\left(\frac{-u_o t}{D_t T}\right).$$

Then

$$\kappa(t) = \frac{D_t C}{u_o}\left(1 - \exp\left(\frac{-u_o t}{D_t T}\right)\right)$$

and by substituting we determine $C = u_c/T$:

$$\psi(t) = \frac{u_c}{D_t}\exp\left(\frac{-u_o t}{D_t T}\right);$$

$$\kappa = \frac{u_c}{u_o}\left(1 - \exp\left(\frac{-u_o t}{D_t T}\right)\right)$$

We obtain the reconstruction time t_{rc} by solving for $\kappa(t_{rc}) = 1$:

$$t_{rc} = -\frac{T D_t}{u_o}\ln\left(1 - \frac{u_o}{u_c}\right).$$

In contrast, the reconstruction proceeds with opportunistic reconstruction only at the rate of

$$\kappa_{opp} = 1 - \exp\left(\frac{-u_o t}{D_t T}\right)$$

Use of a Spare Disk: Example (continued)

We continue our example from the preceding section. The track access time is set at 15 ms consisting of a seek of little less than 5 ms, a short time to the beginning of a block and two latencies. We present an example with only opportunistic reconstruction in Figure 11.9. We assume no temporal locality at all. The graph shows the capacity slowly increasing to 1. The next curve is the utilization of a disk outside the string with the failed disk, which

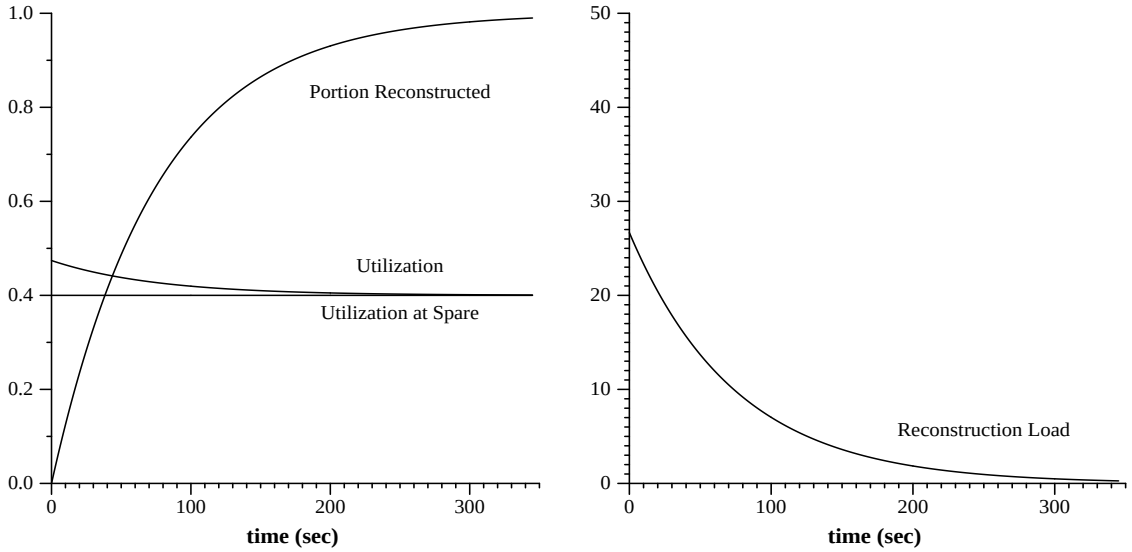


Figure 11.9: Disk Utilization with Opportunistic Load: Shown are protion of reconstruction data on spare, utilization of a disk outside the string with the failed disk, utilization of the spare disk (left) and reconstruction load (right).

decreases from 0.474 to 0.400. The utilization of the spare is constant 0.4. The reconstruction load ζ starts at 0.027 and decreases slowly to zero.

In Figure 11.10 we show a mix of forced and opportunistic reconstruction, that keeps the reconstruction work-load constant. The utilization at the disks outside the string with the failed disk increases slightly due to normal writes involving the the spare. The utilization of the spare doubles, as its additional load is increased by the normal disk load.

Figure 11.11 illustrates the effects of fixing the utilization at the spare at 0.8. In our case, reconstruction is sped up and the utilization at the disks outside the string with the failed disk is increased, though not by the same factor.

Use of a Spare Disk: Write Quorum $w > 2$

All data stored in a reliability group can be accessed through a read at a read quorum of $n - w + 1$ disks. We first discuss the handling of requests directed to the failed disks. A read to the failed disk results in a track read at $n - w + 1$ disks and a reconstruction of the track on the spare. If the track is already reconstructed on the spare, we only read

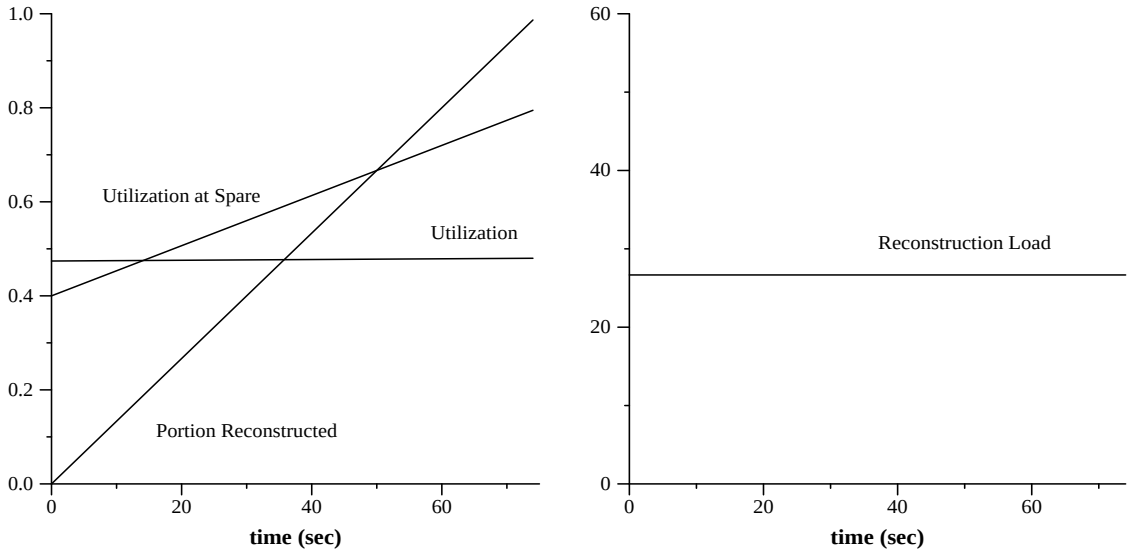


Figure 11.10: Utilization with Constant Reconstruction Load when Reconstructing on Spare in a One-Dimensional Disk Array: Shown are portion of reconstructed data on spare, utilization at a disk outside the string with the failed disk, utilization of the spare disk (left), and reconstruction load (right).

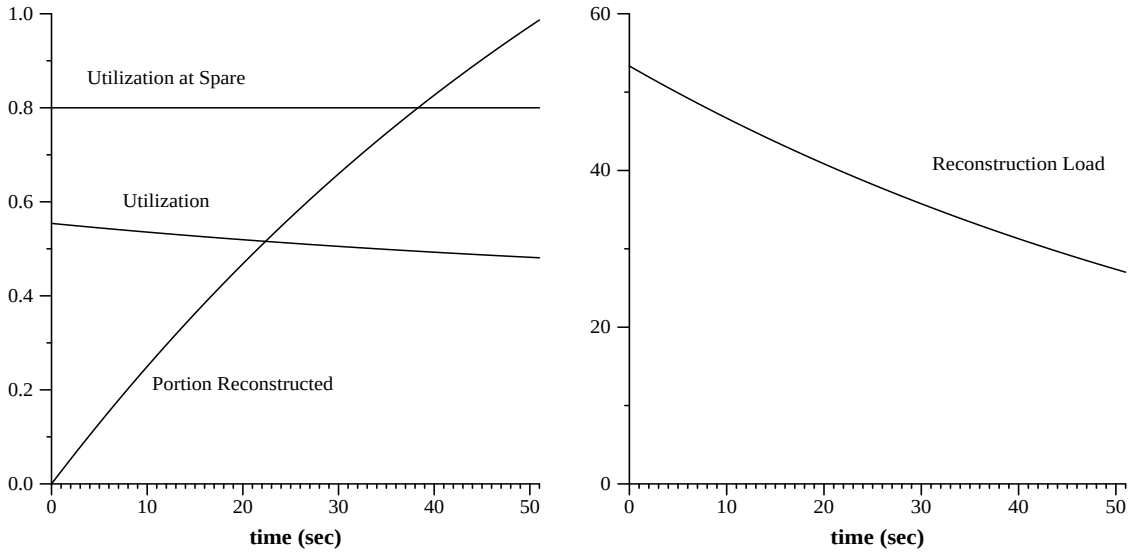


Figure 11.11: Utilization with Constant Spare Utilization when Reconstructing on Spare in a One-Dimensional Disk Array. Shown are portion of reconstructed data on spare, utilization at a disk outside the string with the failed disk, utilization of the spare disk (left), and reconstruction load (right).

it there, of course. We perform a write to the failed disk as the information disk by first reading all existing $n-w$ message tracks. With the data from the write request, we now have enough information to calculate the new check data. We then read the $w-1$ check tracks, but intersperse the read by a write to the check blocks of the updated block, and finally we write the reconstructed track of the failed disk on the spare. Now assume that the write used the failed disk as a check disk. We can save track reads in this case. We write normally to the message disk. This gives us the Δ-value so that we can update the surviving $w-2$ check disks. At the same time, we can read the whole track. Simultaneously, we read the track at all but $w-3$ message disks. We also write the new check blocks in a normal write operation at the $w-2$ remaining check tracks, but reading the whole track. Finally, we now have enough information to reconstruct the track on the spare. We perform $w-1$ write operations (while reading the whole track at the same time) and $n-2w+2$ full track reads. Alternatively, we can perform $n-1$ track read/writes. As the difference in service time between a write and a track read operation is only a single latency, we gain. All forced reconstruction work proceeds with $n-w+1$ track reads.

The pre-failure write load at the disks outside the string with the failed disk will diminish by those writes involving the failed disk as message disk, as far as the spare has not already taken over. This gain is fictitious: the response to these requests leads to a track reconstruction process which uses more resources. The write load to the failed disk is

$$(1-\pi(t))\lambda_w^{old} = \frac{w\rho_w}{\rho_r + w\rho_w}\zeta$$

and each requests involves $w-1$ disks spread over $(n-1)m$ disks. The write load to the failed disk as message disk is $\frac{\rho_w}{\rho_r+w\rho_w}\zeta$ and the loss of writes is $\frac{w-1}{(n-1)m}\frac{\rho_w}{\rho_r+w\rho_w}\zeta$. The new loads are:

$$\begin{aligned}
\lambda_r^{df} &= \lambda_r^{old} \\
\lambda_w^{df} &= \lambda_w^{old} - \frac{\rho_w}{\rho_r + w\rho_w}\frac{(w-1)}{(n-1)m}\zeta \\
\lambda_t^{df} &= \frac{n-w+1}{(n-1)m}\phi + \frac{\rho_r}{\rho_r + w\rho_w}\cdot\frac{n-w+1}{(n-1)m}\zeta \\
&\quad + \frac{\rho_w}{\rho_r + w\rho_w}\cdot\frac{1}{m}\zeta + \frac{(w-1)\rho_w}{\rho_r + w\rho_w}\cdot\frac{n-2w+2}{(n-1)m}\zeta
\end{aligned}$$

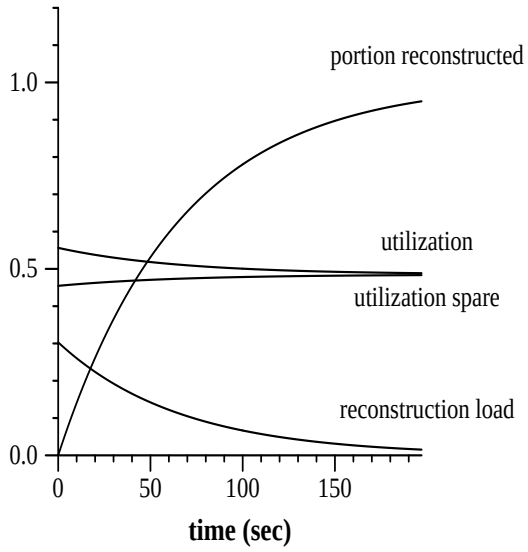


Figure 11.12: Utilization with Opportunistic Load at an MDS based RAID: Shown are Portion of reconstructed data on spare, utilization at a disk outside the string with the failed disk, utilization of the spare disk, and reconstruction load in requests per sec/100.

The load at the spare consists of the reconstruction load and write and read redirects:

$$\begin{aligned} \lambda_r &= \pi(t)\lambda_r^{old} \\ \lambda_w &= \pi(t)\lambda_w^{old} \\ \lambda_t &= \phi + \zeta \\ \zeta &= (1-\pi(t))\lambda^{old} \end{aligned}$$

With opportunistic load only, the utilization at strings outside the failed string will increase somewhat more than without reconstruction on spare. The utilization at the spare is about the same as for all disks in the RAID before the disk failure occurs. (This observation is exactly true only for a specific read to write request ratio but quite a good estimate for reasonable ratios.)

Disk utilization in RAIDs is never very high, so that we can assume that the peak load at the strings without the failed disk is still less than the maximum utilization, at which

we would want to run a disk. We can therefore use the same strategy as before and have the reconstruction load fix the utilization of the spare at this highest reasonable level. In our example below, we choose a utilization of 80% as the maximum. We implement the scheme by measuring the utilization at all disks and depending on the sustained peak utilization issue more or less forced reconstruction orders. This scheme presents the best reconstruction times possible without compromising response times too much. It is self-adjusting as well.

Use of a Spare Disk: Example (continued):

We continue our previous example for an MDS based RAID with 5 reliability groups and 11 strings. We use the same RAID load, so that the baseline utilization of the disks is about 12% higher than in the one-dimensional RAID.

In Figure 11.12 we show the equivalent situation to the one described in Figure 11.9. While the utilization increase at disks outside the string with the failed disk is considerable ($> 20\%$) it slowly returns to the baseline utilization, as the spare is reconfigured. Eventually, we have to use forced reconstruction to finish the task. The amount of reconstruction is very dependent on the temporal locality behavior, and a higher locality than in our memory-less sample situation will lead to higher initial reconstruction and higher spare load. Real disk access behavior will also see large areas of the disk not touched for a lengthy period.

Our second Figure (11.13) shows the utilization of disks if the reconstruction load is kept constant. Initially, the utilization at the disks outside the string with the failed disk is higher, but soon the spare utilization surpasses is to double as the reconstruction process terminates.

The third Figure (11.14) depicts the effects of the scheme that keeps the peak utilization, in this case at the spare only, at 80% utilization. This affords us the fastest reconstruction time.

Use of Spare Disk: String Failure:

In the case of a string failure, the spare disks cannot reconstruct the contents of the string. We do not explore the performance as it will not be different from the one in a RAID without any spares at all.

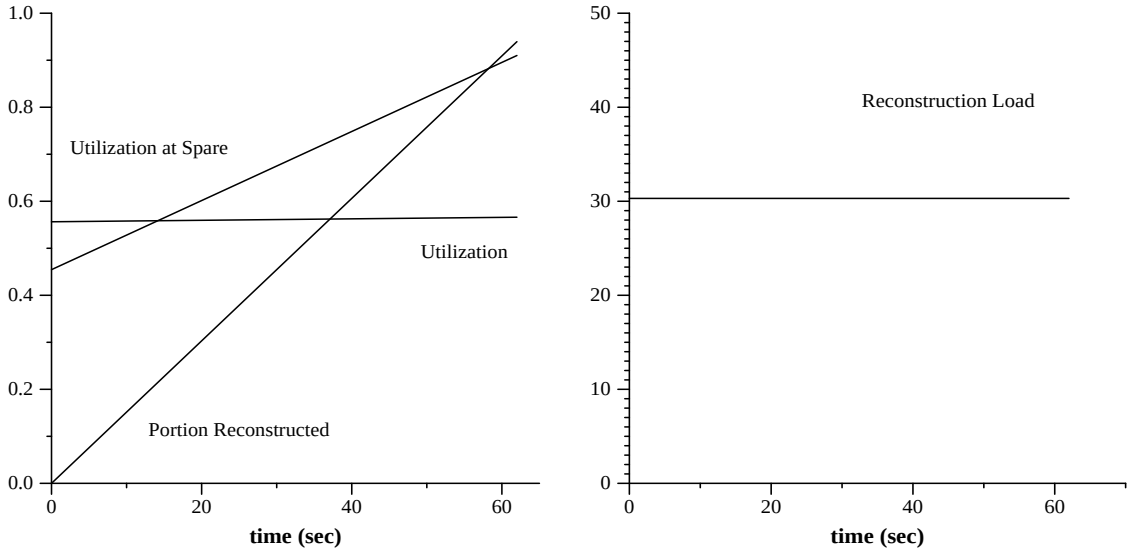


Figure 11.13: Utilization with Constant Reconstruction Load at an MDS based RAID while Reconstructing on Spare.

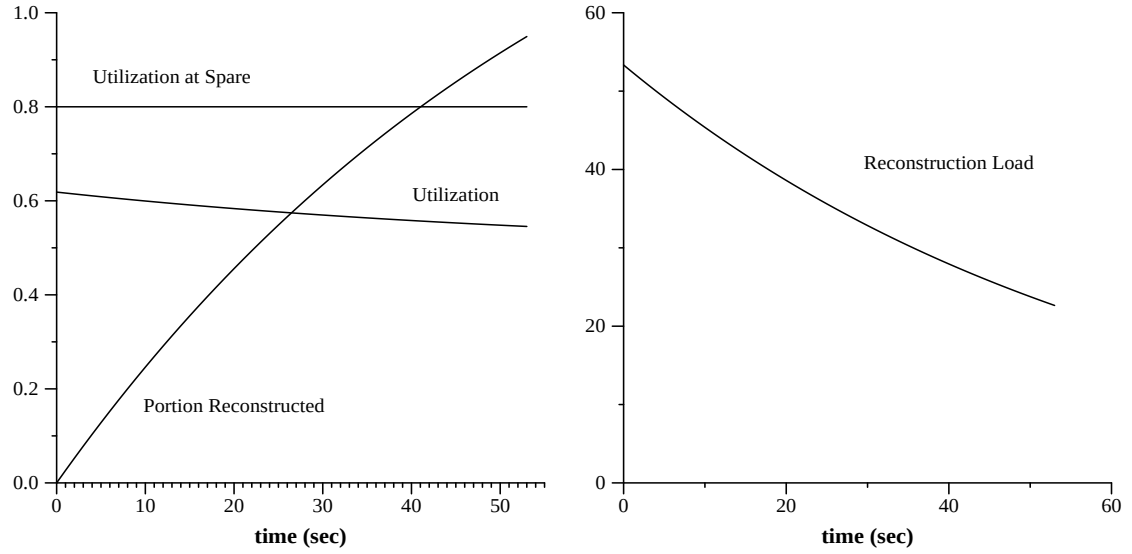


Figure 11.14: Utilization with Constant Spare Utilization at an MDS based RAID. Reconstruction load is given in requests per millisecond.

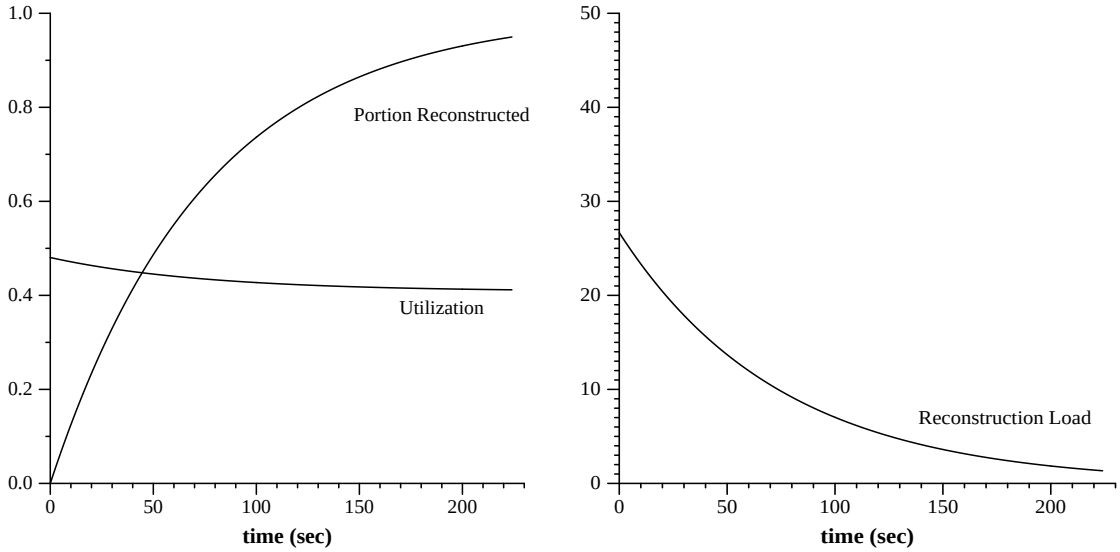


Figure 11.15: Utilization with Opportunistic Reconstruction only in the NDS Scheme.

11.2.4 Naive Distributed Sparing

In distributed sparing, one or two disks worth of free space are distributed over all the disks in the RAID. The naive approach transfers the contents of a failed disk onto spare space at reconfiguration. In contrast, safe distributed sparing treats these writes as RAID writes and updates check information. Naive Distributed Sparing does not protect well against the effects of string failure. It makes excellent use of the spare space, when it is not in use to replace failed disk(s), by spreading the baseline load out over more disks than a scheme with a string of spares. As we will see, this behavior remains true for the performance with aggressive reconstruction of data on a just failed disk.

The analytical treatment of Naive Distributed Sparing (NDS) is easy. We divide the utilization and the loads at the spare disk among all surviving disks. Distributive sparing sets certain tracks apart as spare space. In NDS, this diminishes the number of tracks in some reliability group. If the equivalent of s disks (typically $s = 1$ or $s = 2$) are used for sparing then the average reliability group has $\frac{s(n-1)+(m-s)n}{m}$ tracks. The treatment of string failure is superfluous, as we do not have a whole spare string.

NDS: Write Quorum $w = 2$

For a write quorum of two, a reconstruction operation accesses

$$\sigma = \frac{s(n-2) + (m-s)(n-1)}{m}$$

disks besides the one written. We have:

$$\begin{aligned}
\lambda_r^{df} &= \lambda_r^{old} + \frac{\pi(t)}{nm-1}\lambda_r^{old} \\
\lambda_w^{df} &= \lambda_w^{old} - \frac{1-\pi(t)}{(n-1)m}\lambda_w^{old} + \frac{\pi(t)}{nm-1}\lambda_w^{old} \\
\lambda_t^{df} &= \frac{\sigma}{(n-1)m}(\phi+\zeta) + \frac{\phi+\zeta}{nm-1} \\
\zeta &= (1-\pi(t))\lambda^{old}
\end{aligned}$$

The load at the disks in the same string as the failed disk now increases insignificantly to

$$\begin{aligned}
\lambda_r^{1} &= \lambda_r^{old} + \frac{\pi(t)}{nm-1}\lambda_r^{old} \\
\lambda_w^{1} &= \lambda_w^{old} - + \frac{\pi(t)}{nm-1}\lambda_w^{old} \\
\lambda_t^{1} &= \frac{\phi+\zeta}{nm-1} \\
\zeta &= (1-\pi(t))\lambda^{old}
\end{aligned}$$

Distributed Sparing makes comparison to other RAID schemes difficult, because the RAID storage capacity is different. We assume here, that the disk utilization without failure is left the same. With opportunistic reconstruction only, the peak utilization increases by 20.115% to about 0.48046Λ compared to a 12.593% increase to 0.450370Λ when no spares are being used. The difference between the utilizations in both cases can be use to adjust to very high baseline utilizations, when disallowing reconstruction permits us to run the RAID in stable state.

The best reconstruction times given a maximal allowable utilization (which immediately translates into response times) sets the disk utilization constant, until the reconstruction process is terminated and the utilizations go back to normal.

NDS: Example (continued)

We continue the running example: In Figure 11.15 we show the utilization effects of NDS using only opportunistic reconstruction. Figures 11.16 show the reconstruction process, if the utilization at the disks outside the string with the failed disks is kept constant.

NDS: Write Quorum $w > 2$

We can modify the load formulae for the RAID with spares in this case as well. The average reliability group consists of $\sigma = \frac{s(n-2)+(m-s)(n-1)}{m}$ tracks. The other source of changes is the distribution of the spare load on all remaining disks in the RAID.

$$
\begin{aligned}
\lambda_r^{df} &= \lambda_r^{old} + \frac{\pi(t)}{nm-1}\lambda_r^{old} \\
\lambda_w^{df} &= \lambda_w^{old} - \frac{\rho_w}{\rho_r + w\rho_w}\frac{(w-1)}{(n-1)m}\zeta + \frac{\pi(t)}{nm-1}\lambda_w^{old} \\
\lambda_t^{df} &= \frac{\sigma+2-w}{(n-1)m}\phi + \frac{\rho_r}{\rho_r + w\rho_w}\cdot\frac{\sigma+2-w}{(n-1)m}\zeta \\
&\quad + \frac{\rho_w}{\rho_r + w\rho_w}\cdot\frac{\sigma}{(n-1)m}\zeta + \frac{(w-1)\rho_w}{\rho_r + w\rho_w}\cdot\frac{\sigma-2w+3}{(n-1)m}\zeta \\
&\quad + \frac{\phi+\zeta}{nm-1}
\end{aligned}
$$

Because the reconstruction load is distributed over many disks, NDS offers the shortest possible reconstruction times. On the other hand, even requests not directed to the failed disk will suffer noticeable degeneration in the response times.

NDS: Example (continued):

We present in Figures 11.17 and 11.18 the reconstruction process for our example MDS array.

11.2.5 Safe Distributed Sparing

Safe Distributed Sparing (SDS) safeguards the spare space by treating the reconstruction process as RAID writes. As we have seen, SDS can improve the reliability considerably.

In the SDS as in the NDS scheme, the equivalent space of s (typically one or two) disks is set apart as spare tracks. These tracks are uniformly distributed through the whole

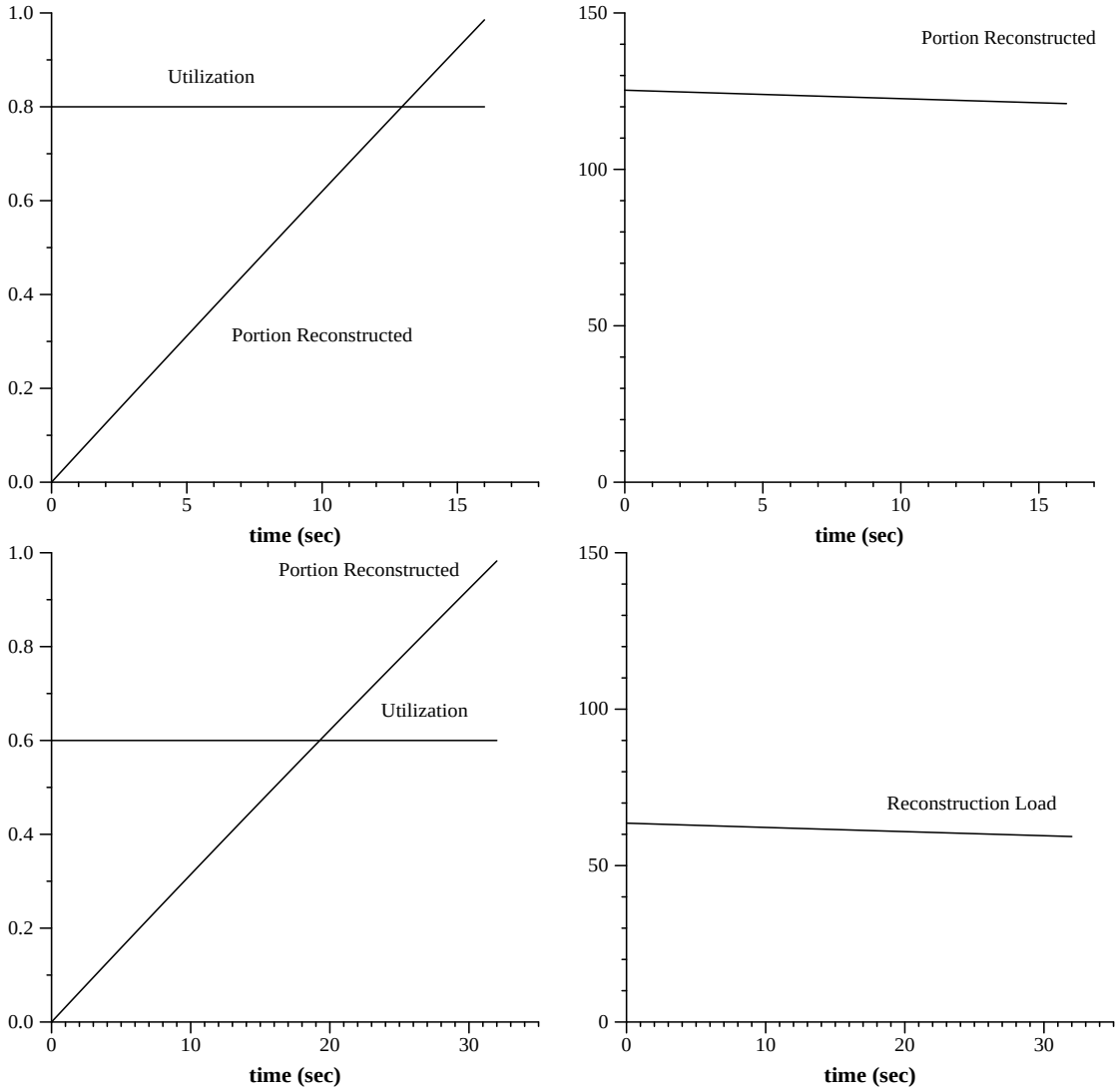


Figure 11.16: Effects of Constant Utilization in the NDS Scheme.

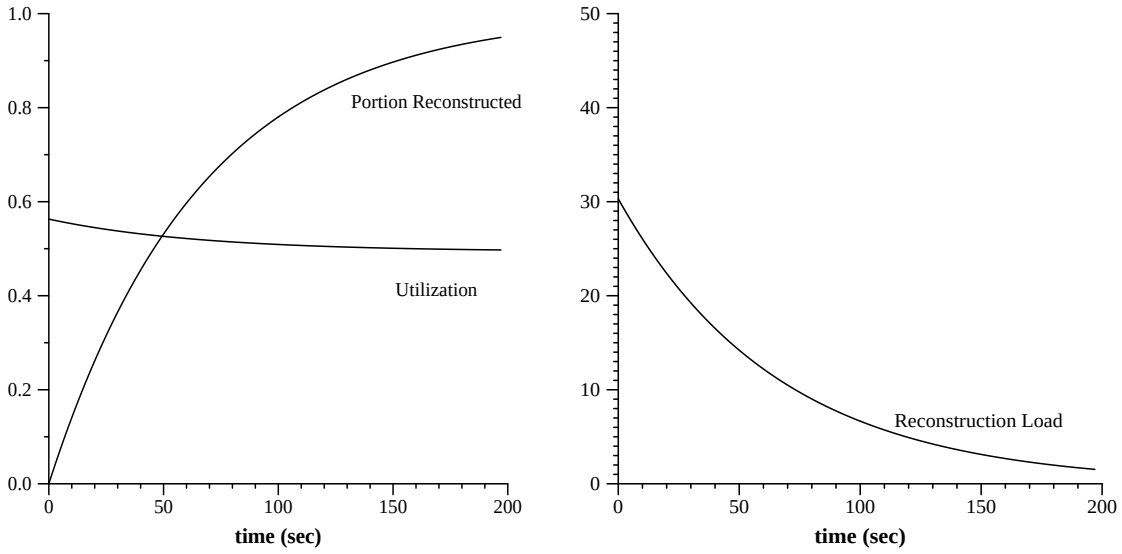


Figure 11.17: Opportunistic Reconstruction in the NDS Scheme for an MDS based RAID.

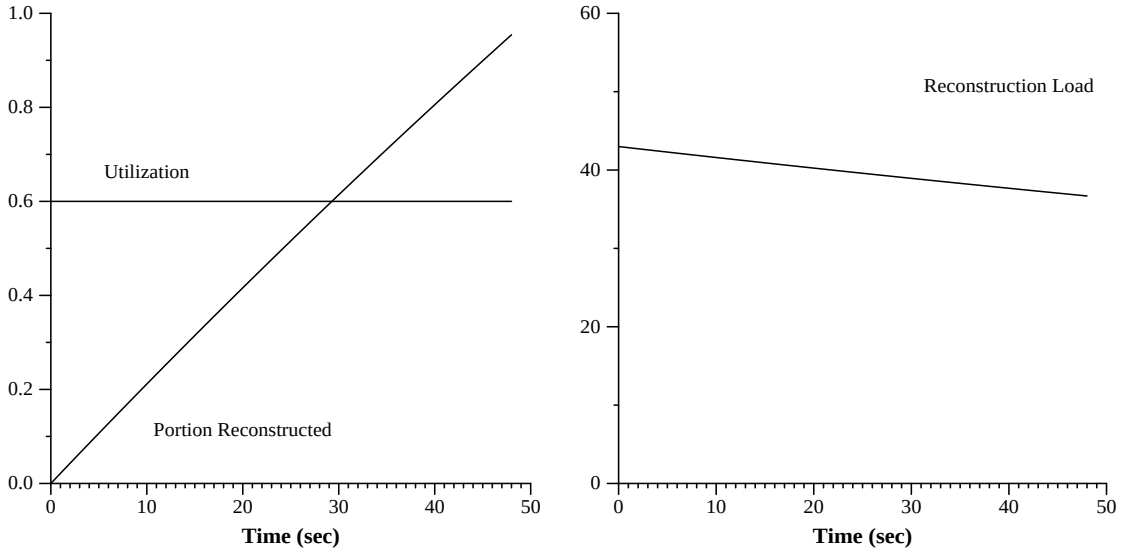


Figure 11.18: Constant Peak Utilization in the NDS Scheme for an MDS based RAID.(Reconstruction Load is given in requests per 10 ms.)

RAID. If after a disk failure and subsequent reconstruction on the spare space, a string fails, which did not contain the failed disk, the NDS scheme (with $w = 2$) will suffer data-loss, because a spare track (now containing reconstructed data) and another track in the same reliability group, both located on the failed string, are lost at the same time. This is not possible in SDS, as the spare track is treated like a track in another reliability group, which then can be reconstructed. (Of course, a RAID which needs multiple reconstruction steps might not be able to handle all user requests.)

The drawback to the reliability gain is the additional reconstruction work. It is possible to limit this by using a two-step procedure, in which the reconstruction takes place as in the NDS scheme and only after this step is finished, (as we have seen possibly in a mere 30 seconds) is followed by an update of the checks of those reliability groups, that host a now used spare as a "guest". In contrast, we analyze a one step procedure.

In Table 3.14 we had given an example of distributed sparing before and after disk failure. As can be seen from this small example, we need to perform an additional write to a check track, if the spare space is in a different reliability group than the track under reconstruction. In contrast, if the failed disk is in the same reliability group than the spare track for this particular track, then the check information need not be changed. With probability $p = 1 - 1/m$ we have to change a check track. The procedure is very close to that of a normal update. We need to read the old track data on the spare (which we cannot assume to be zero) and read and write the check track. We can shift work from the reconstruction period immediately after the disk failure to the repair process: The check data never depends on the data on the spare track. They are assumed to be zero. When this spare track is starting to get used, during the reconstruction period, the Δ-values of the reconstruction data are the reconstruction data themselves and are embedded into the new check data. During the repair phase, when the spare track data are placed on the replacement disk for the failed disk, we change the check track value back, so that it does not incorporate the spare track data. We then assume that the spare track is zero again. By modifying the size of the reliability group, we have shifted utilization from the reconstruction phase to the repair phase, when the utilization increase can be kept as small as desired. The only disadvantage of this scheme is the need for the RAID controller to distinguish between reliability groups of different size.

To determine the loads and hence the utilization of the disks outside the string with the failed disk, we just add the additional track reads followed by a track writes to the formulae for NDS. We denote the new category of loads by λ_{tw}. The service time consists of 2 latencies more than for the track read. In our running example, the service time D_{tw} for the track writes is 25 ms.

SDS: Write Quorum $w = 2$

For a write quorum of two we have:

$$\begin{aligned}
\lambda_r^{df} &= \lambda_r^{old} + \frac{\pi(t)}{nm-1}\lambda_r^{old} \\
\lambda_w^{df} &= \lambda_w^{old} - \frac{1-\pi(t)}{(n-1)m}\lambda_w^{old} + \frac{\pi(t)}{nm-1}\lambda_w^{old} \\
\lambda_t^{df} &= \frac{\sigma}{(n-1)m}(\phi+\zeta) + \frac{1}{nm-1}(\phi+\zeta) \\
\lambda_{tw}^{df} &= \frac{1}{nm-1}\frac{m-1}{m}(\phi+\zeta) \\
\zeta &= (1-\pi(t))\lambda^{old}
\end{aligned}$$

Using opportunistic reconstruction, the utilization at the disks outside the string with the failed disk increases to 0.491344Λ (a 22.84% increase over the baseline utilization,) which is only insignificantly larger than in the NDS scheme. Consequentially, the reconstruction times using the constant utilization scheme are only insignificantly larger than the corresponding times for NDS. This observation imply that SDS combines good reliability with good performance in the normal case as well as while reconstructing a failed disk on spare space. In our opinion, it supplements hardening strings in achieving a very attractive RAID organization. We give details of the reconstruction process in Figures 11.19 and 11.20.

SDS: Write Quorum $w > 2$

In case that the write quorum exceeds two, we have:

$$\lambda_r^{df} = \lambda_r^{old} + \frac{\pi(t)}{nm-1}\lambda_r^{old}$$

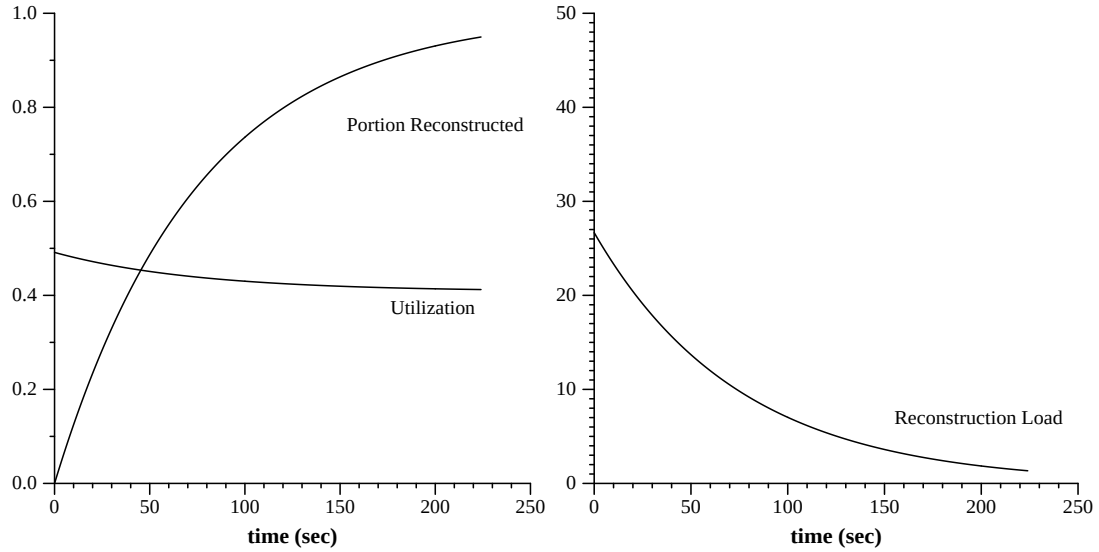


Figure 11.19: Opportunistic Reconstruction in the SDS Scheme for an one dimensional RAID.

$$
\begin{aligned}
\lambda_w^{df} &= \lambda_w^{old} - \frac{\rho_w}{\rho_r + w\rho_w}\frac{(w-1)}{(n-1)m}\zeta\frac{\pi(t)}{nm-1}\lambda_w^{old} \\
\lambda_t^{df} &= \frac{\sigma+2-w}{(n-1)m}\phi + \frac{\rho_r}{\rho_r + w\rho_w}\cdot\frac{\sigma+2-w}{(n-1)m}\zeta \\
&\quad + \frac{\rho_w}{\rho_r + w\rho_w}\frac{\sigma}{(n-1)m}\zeta + \frac{(w-1)\rho_w}{\rho_r + w\rho_w}\frac{\sigma+(3-2w)}{(n-1)m}\zeta + \frac{\pi(t)}{nm-1}(\phi+\zeta) \\
\lambda_{tw}^{df} &= \frac{1}{nm-1}\frac{m-1}{m}(\phi+\zeta)
\end{aligned}
$$

The peak utilization under SDS using opportunistic reconstruction reaches 0.566195Λ. We present the case in Figure 11.21.

11.2.6 Distributed Sparing of a String

Distributed sparing of a whole string is a performance boosting scheme due to Menon and Mattson ([22]. In it, a string's worth of spare space is equally distributed throughout the disk array. Consequentially, disk utilization is diminished, because the load is spread over more disks. To be more precise, the utilization of a RAID with n reliability groups in the absence of failures is diminished by a factor of $1/n$ and consequentially, the response time, by a factor of $\frac{U}{(1-U)n.}$

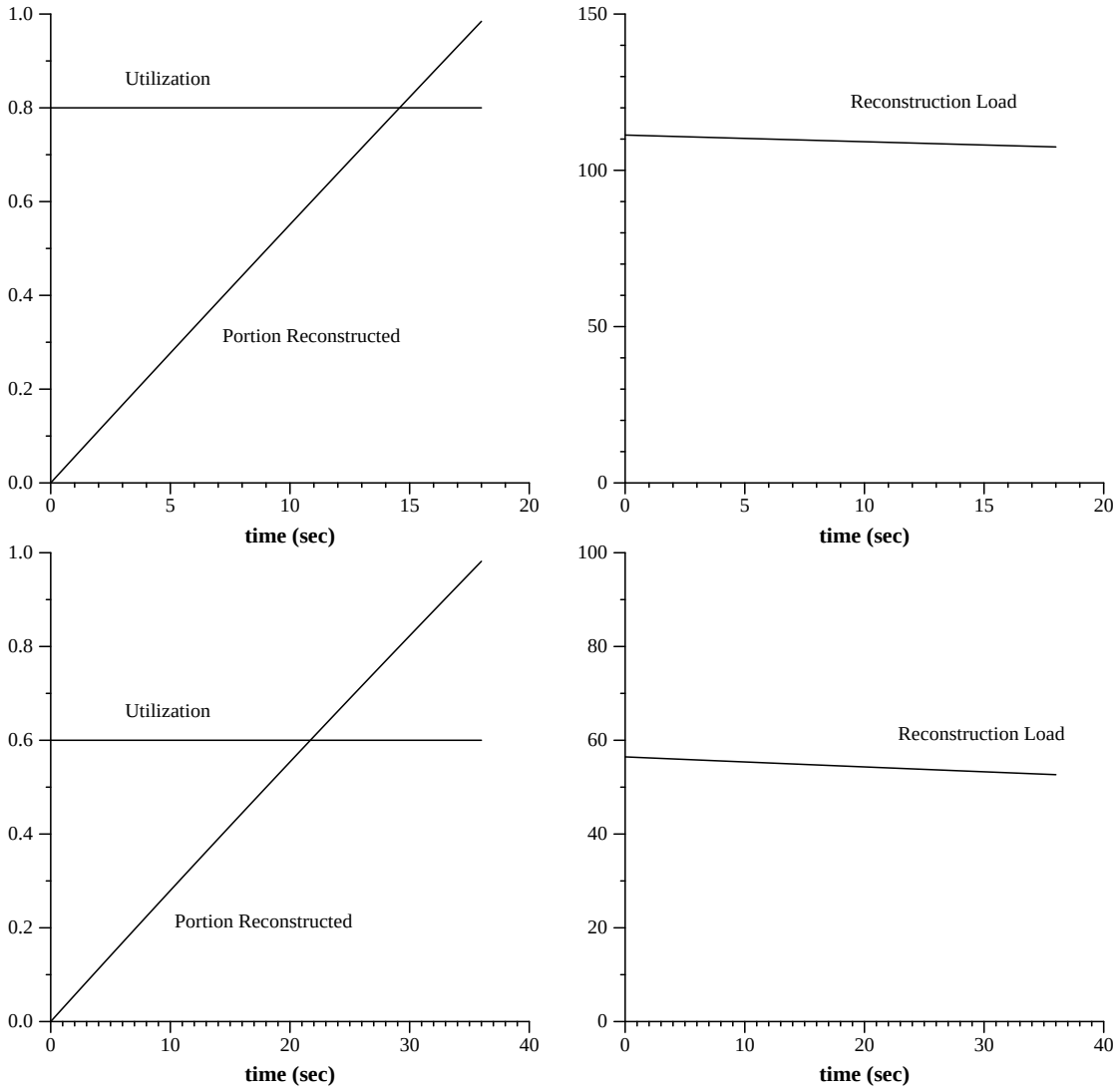


Figure 11.20: Constant Peak Utilization in the SDS Scheme for an one dimensional RAID. (Reconstruction Load is given in requests per 10 ms.)

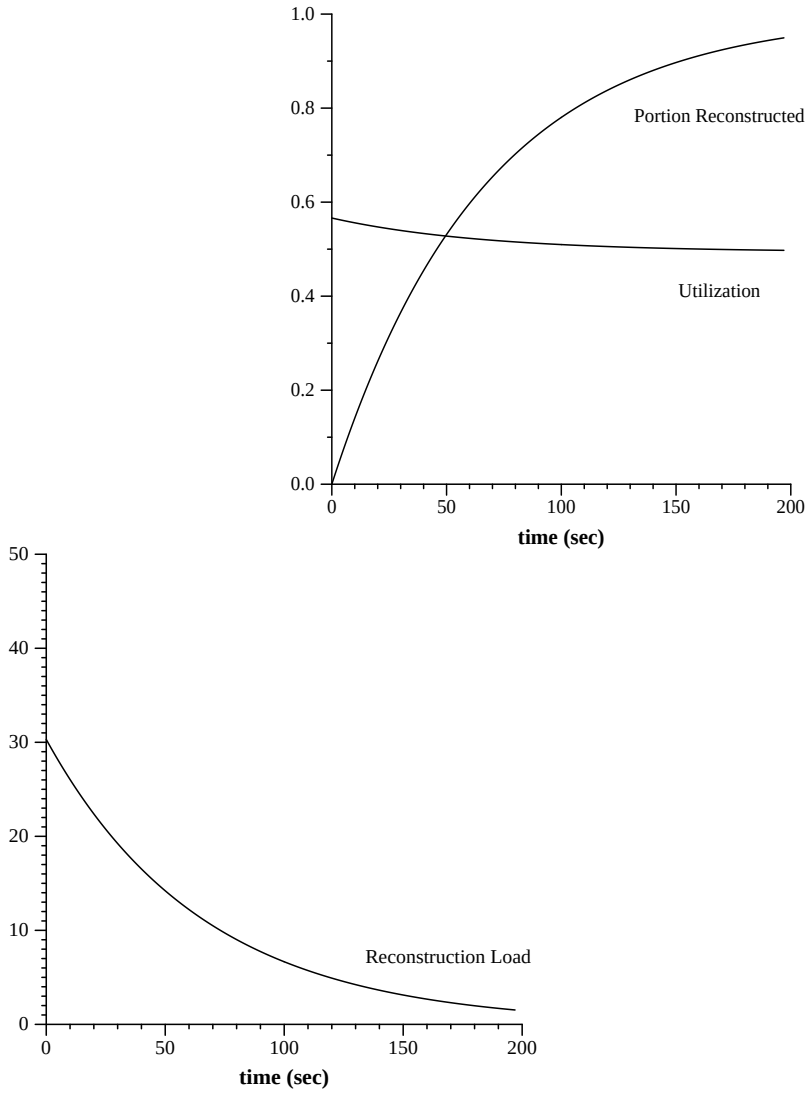


Figure 11.21: Opportunistic Reconstruction in the SDS Scheme for an MDS code based RAID. (Reconstruction Load is given in requests per 10 ms.)

We can embed Distributed String Sparing in ACATS. One of the virtual string addresses refers to the the spare string. Because the string addresses are permuted, the spare string will end up distributed equally among all actual strings. The in-string disk address permutations will also assign a given virtual spare track with equal probability to all reliability groups. In contrast to NDS a string failure following a disk failure cannot lead to data loss.

The reconstruction work in the disk failure case is equally distributed among all non-failed disks. Loads and utilization are consequentially the same as in the NDS scheme. For a write quorum of $w = 2$ we obtain:

$$\begin{aligned}
\lambda_r^{df} &= \lambda_r^{old} + \frac{\pi}{nm-1}\lambda^r \\
\lambda_w^{df} &= \lambda_w^{old} - \frac{1-\pi}{(n-1)m} \cdot \lambda_w^{old} + \frac{\pi}{nm-1} \cdot \lambda_w^{old} \\
\lambda_t^{df} &= \frac{n-2}{(n-1)m} \cdot (\phi + \zeta) + \frac{\phi + \zeta}{nm-1} \\
\zeta &= (1 - \pi(t))\lambda^{old}
\end{aligned}$$

We now treat the case of a string failure.

Distributed String Sparing: Write Quorum $w = 2$

If teh write quorum is two, the reconfiguration process proceeds essentially as outlined in Section 11.2.3. Because data from m disks have to be reconstructed, the total reconstruction load for string failure is magnified by this factor over the disk failure reconstruction load. In more detail: The read load at a surviving disk is increased by the reads directed to spare tracks; this increase becomes permanent after the reconstruction process is finished and reflects the lost savings of Distributed Sparing. The write load at a surviving disks is diminished by writes involving by the failed string, that can not be handled by the spare disks, and increased by the load to the spare string. The latter increase is permanent. We have a track access load consisting of reads to reconstruct data and writes of these data. As a formula, we have:

$$
\begin{aligned}
\lambda_r^{ds} &= \left(1 + \frac{\pi(t)}{n-1}\right)\lambda_r^{old} \\
\lambda_w^{ds} &= (1 - \frac{1-\pi(t)}{n-1} + \frac{\pi(t)}{n-1})\lambda_w^{old} \\
&= \frac{n-2+2\pi(t)}{n-1}\lambda_w^{old} \\
\lambda_t^{ds} &= \left(1 + \frac{1}{n-1}\right)(\phi + \zeta) \\
\zeta &= (1-\pi(t))\lambda^{old}
\end{aligned}
$$

Distributed String Sparing: Write Quorum $w > 2$

With analogue justification as before we obtain:

$$
\begin{aligned}
\lambda_r^{ds} &= \left(1 + \frac{\pi(t)}{n-1}\right)\lambda_r^{old} \\
\lambda_w^{ds} &= \lambda_w^{old} - \frac{1-\pi(t)}{n-1}\lambda_w^{old} + \frac{\pi(t)}{n-1}\lambda_w^{old} \\
&= \frac{n-2+2\pi(t)}{n-1}\lambda_w^{old} \\
\lambda_t^{ds} &= \frac{n-w+2}{n-1}\phi + \frac{\rho_r}{\rho_r + w\rho_w}\cdot\frac{n-w+1}{n-1}\zeta + \frac{\rho_w}{\rho_r + w\rho_w}\zeta \\
&\quad + \frac{(w-1)\rho_w}{\rho_r + w\rho_w}\frac{n-2w+2}{n-1}\zeta + \frac{1}{n-1}\zeta
\end{aligned}
$$

Again, we need to notice that some of the increases are permanent and represent the lost savings from the distributed sparing scheme.

Distributed String Sparing: Example (continued)

We illustrate the reconstruction behavior in the string failure case in Figures 11.22 to 11.25. We can observe that the disk utilization is much higher than in the disk failures cases. If the baseline utilization is high, it can happen that the RAID has not enough capacity to handle the reconstruction. If we do not reconstruct at all, we put less load on the disks, but we will not see the performance getting better.

In our examples, the baseline loads are not high enough to prohibit reconstruction. However, as the utilization is almost doubled in the beginning of a reconstruction phase, almost only opportunistic reconstruction takes place and the reconstruction times are higher than in the disk failure cases.

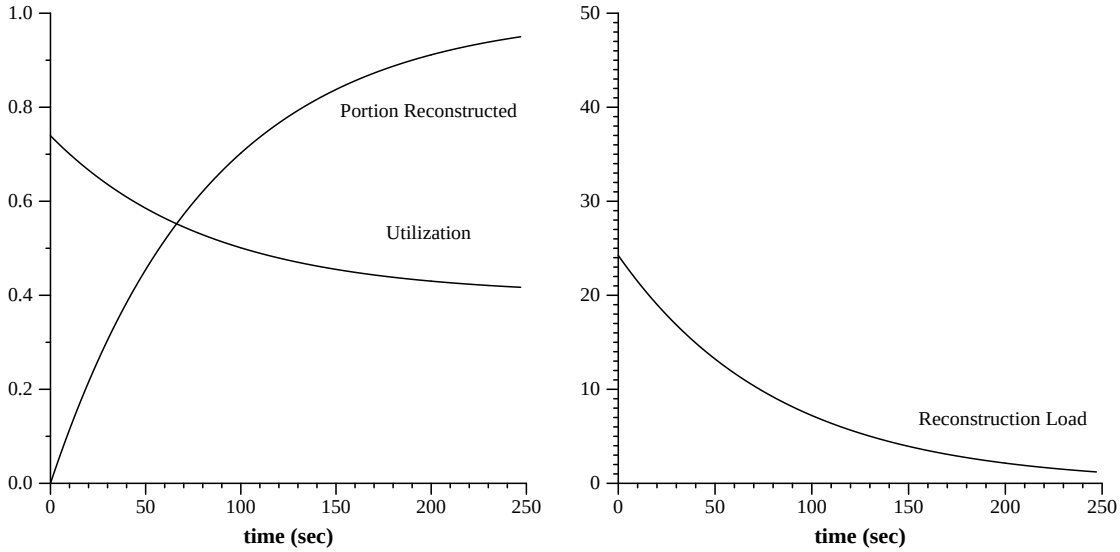


Figure 11.22: Opportunistic Reconstruction in the Distributed Sparing Scheme for a one-dimensional RAID after String Failure.

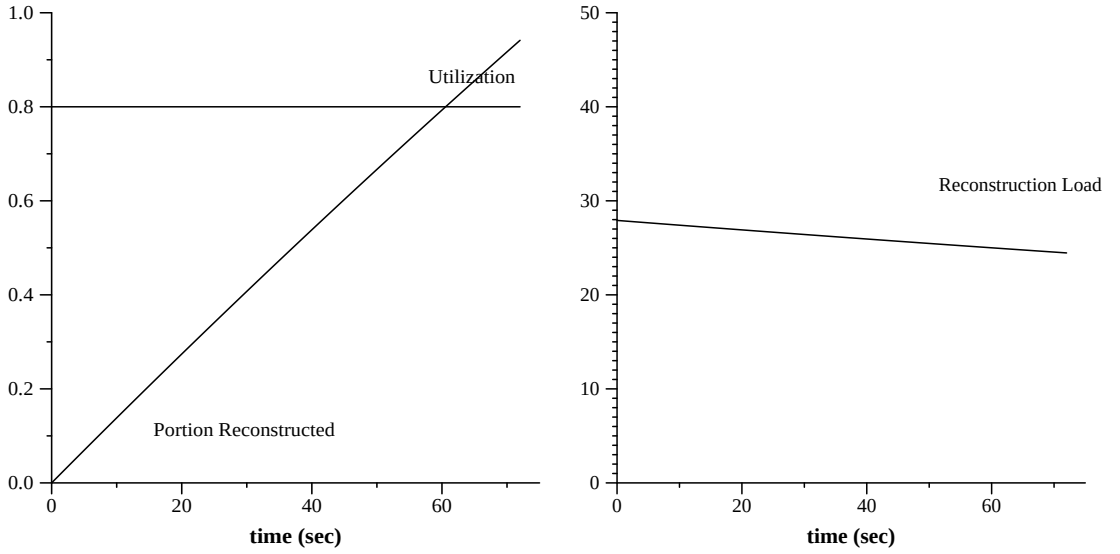


Figure 11.23: Reconstruction in the Distributed Sparing Scheme for a one-dimensional RAID with a Target Disk Utilization of 80% after String Failure.

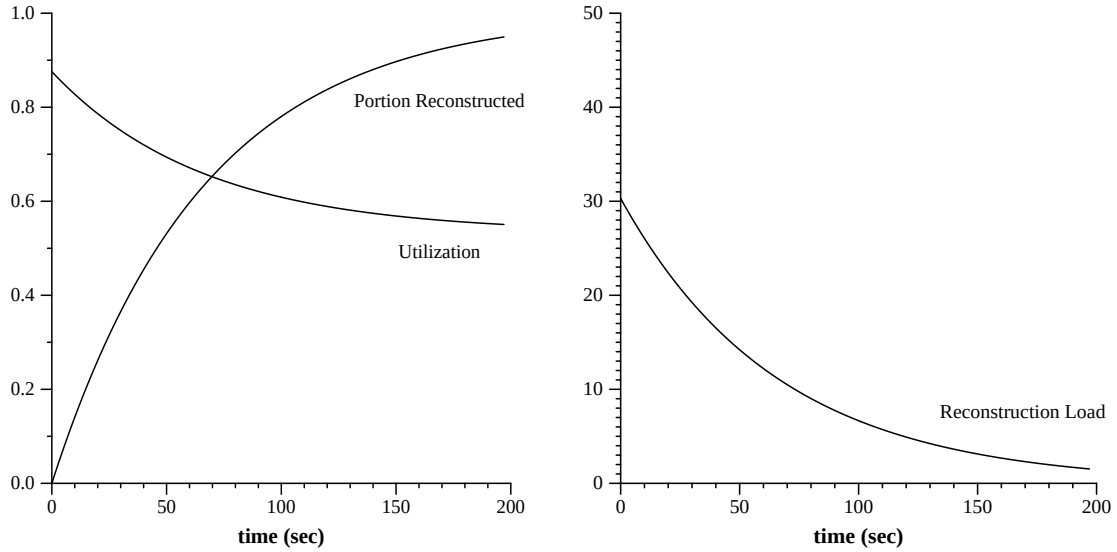


Figure 11.24: Opportunistic Reconstruction in the Distributed Sparing Scheme for an MDS code based RAID.

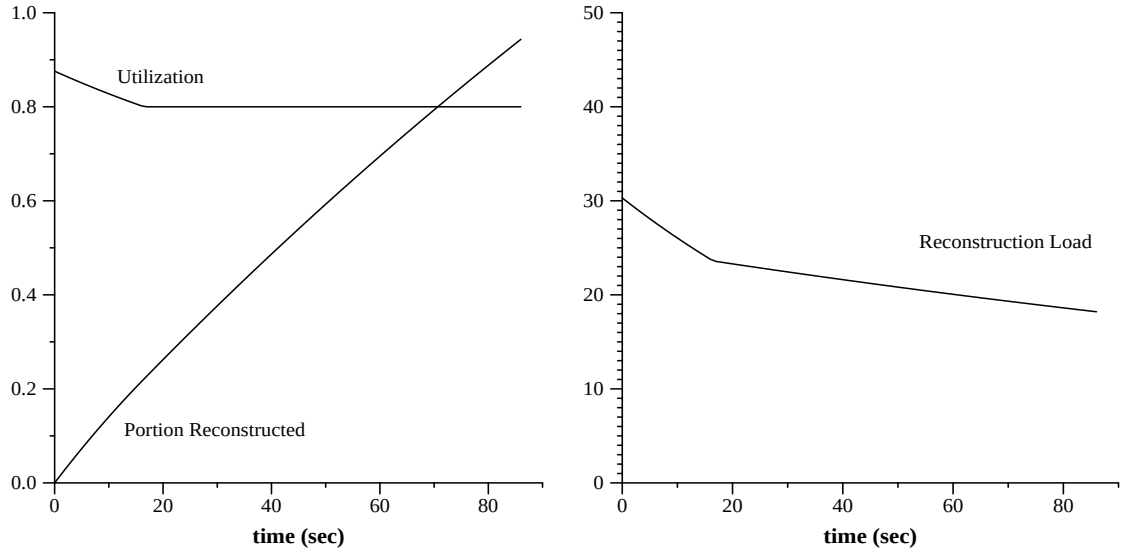


Figure 11.25: Reconstruction in the Distributed Sparing Scheme for an MDS code based RAID with a Target Disk Utilization of 80%.

11.2.7 Comparison of an MDS Based RAID with a Level 5 RAID with Distributed Sparing

It is interesting to compare the performance of two RAIDs with the same hardware use and the same storage capacity. We compare the Level 5 RAID with one distributed spare string with an MDS RAID with two check strings. Both RAID organizations employ 55 disks organized in 11 strings. The storage capacity is the one of 45 disks. In case of a failure, the Level 5 uses reconstruction and the MDS RAID uses reconfiguration. The reconfiguration process for the MDS RAID is cut back as far as possible. We reconfigure only information data on one of the check disks; hence, after reconfiguration an afflicted reliability group will still contain two different kinds of check disks. This version of reconfiguration saves labor over the one we explained earlier.

In the fault-free case, the disk utilization of the Level 5 RAID is far better than the one of the MDS RAID, namely 0.363636Λ as opposed to 0.484848Λ for the MDS RAID, where Λ denotes the RAID load. After a disk failure, both RAID organizations suffer an unavoidable utilization peak of 0.438249Λ for the Level 5 RAID and 0.518989Λ for the MDS RAID, which is caused by opportunistic reconstruction or reconfiguration respectively only. After the reconstruction process the disk uitlization in the Level 5 RAID is increased because the performance benefits of the distributed spare string are not fully available. The final disk utilization for the MDS RAID is lowered, because one reliability group uses only accesses two disks as opposed to three disks previously during a write operation. The effects of a string failure are more extensive. Here we see, that the lighter reconfiguration process at the MDS RAID leads to lower disk utilization, namely 0.672727Λ compared to 0.703030Λ for the Level 5 RAID. The final utilization for both RAIDs are 0.40000Λ.

If we are interested in RAID performance guarantees and want to extend these guarantees to the string failure case, we conclude, that actually the MDS RAID has better performance. This conclusion is in spite of the one third higher disk utilization of the MDS RAID in the fault-free case.

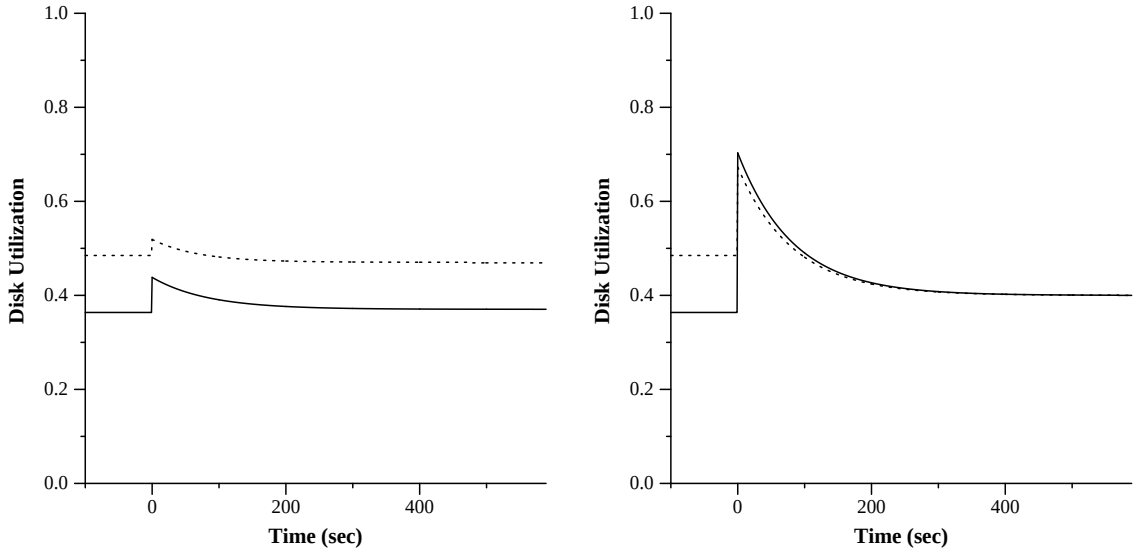


Figure 11.26: Disk Utilization for a Level 5 RAID with Distributed Spare (solid line) and an MDS RAID (dotted line) in the Disk Failure Case (left) and the String Failure Case (right).

11.3 Classic Level 5 RAIDs

In one way or other, we have already done most of the analytical work for this organization. Because in a classic Level 5 RAID the reliability groups are fixed and what happens in one does not influence performance at another reliability group, we can consider a classic Level 5 RAID for all our practical purposes as being an ensemble of m mini-RAIDs each consisting of a single reliability group. The peak utilization in a classic Level 5 RAID does not depend on the kind of component that failed, but the number of affected disks does.

11.3.1 No Spares

We can apply the results for string failure from Section 11.2.1 to the classic Level 5 RAID for both disk and string failure. In the former case, just set $m = 1$. There is no performance gain for ACATS in the string failure case, but a considerable one for the disk failure case. If RAID reconfiguration is used, the performance benefits in case of a disk

failure are limited in a classic Level 5 RAID to the one reliability group affected and not spread, as in ACATS over most disks.

11.3.2 Naive Distributed Sparing

In Naive Distributed Sparing a space amounting to s disks (typically $s = 1$ or $s = 2$) is set aside to provide room for reconstruction of data on a failed disk. Even though the classic Level 5 RAID features fixed reliability groups, this spare space is evenly distributed among all disks. A given reliability group consists on average of $\frac{s(n-1)+(m-s)n}{m}$ tracks.

Classic NDS: Write Quorum $w = 2$

A reconstruction operation needs to access

$$\sigma = \frac{s(n-2)+(m-s)(n-1)}{m}$$

tracks besides the one to be written. We can calculate the load at the surviving disks in the same reliability group with the failed one:

$$\begin{aligned}
\lambda_r^{df} &= \lambda_r^{old} + \frac{\pi(t)}{nm-1}\lambda_r^{old} \\
\lambda_w^{df} &= \lambda_w^{old} - \frac{1-\pi(t)}{n-1}\lambda_w^{old} + \frac{\pi(t)}{nm-1}\lambda_w^{old} \\
\lambda_t^{df} &= \frac{\sigma}{n-1}(\phi+\zeta) + \frac{\phi+\zeta}{nm-1} \\
\zeta &= (1-\pi(t))\lambda^{old}
\end{aligned}$$

The load at the other disks increases insignificantly to

$$\begin{aligned}
\lambda_w^{df'} &= \lambda_r^{old} + \frac{\pi(t)}{nm-1}\lambda_r^{old} \\
\lambda_w^{df'} &= \lambda_w^{old} + \frac{\pi(t)}{nm-1}\lambda_w^{old} \\
\lambda_t^{df'} &= \frac{\zeta}{nm-1}
\end{aligned}$$

The utilization at the disks in the same reliability group almost doubles in the beginning, but then returns to the a level that is only slightly increased over the baseline utilization and coincides with the utilization throughout the RAID. Because the peak utilization at the disks in the same reliability group is high, the reconstruction times are much higher.

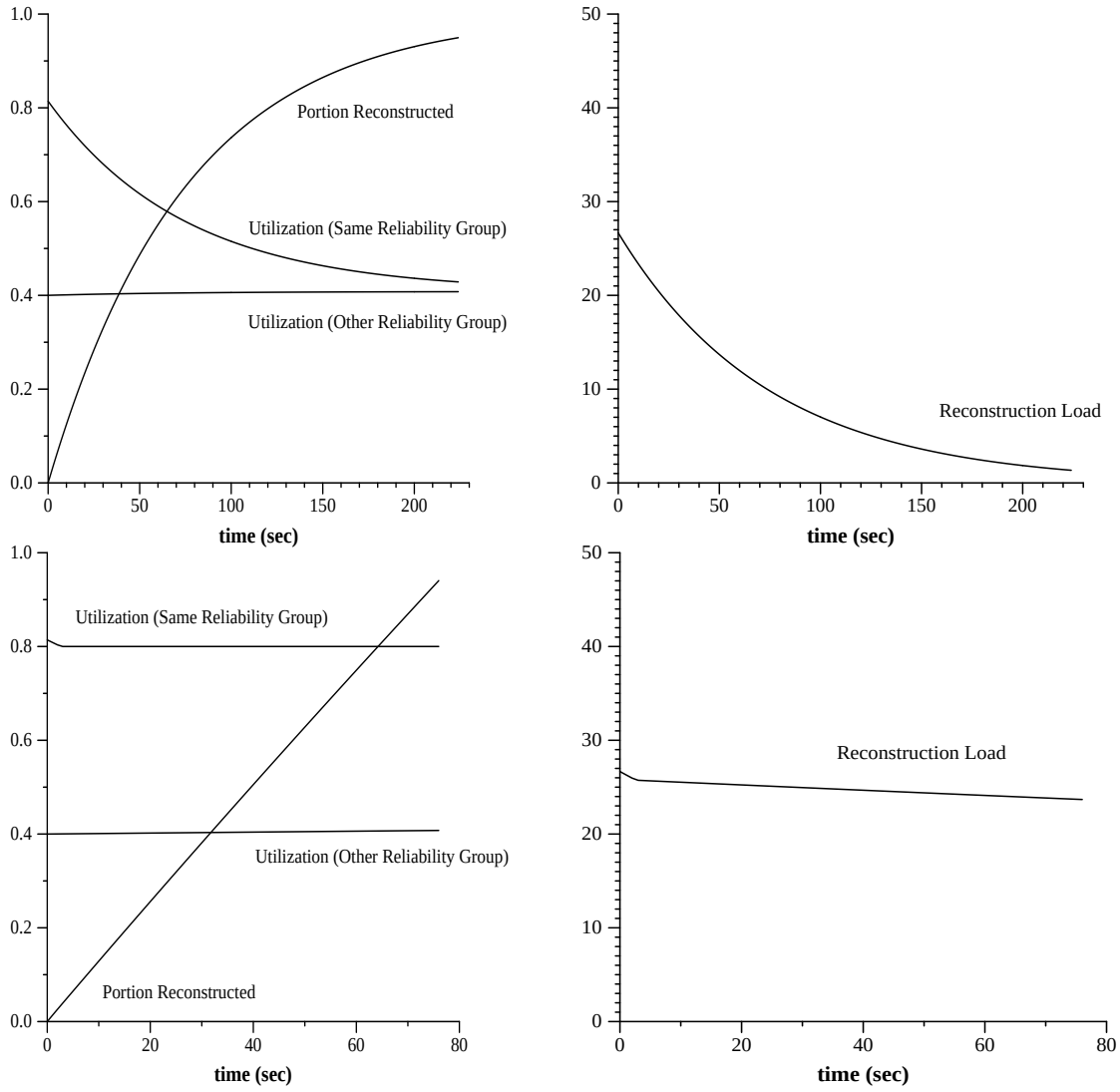


Figure 11.27: Disk Utilization in the NDS scheme for a Classic Level 5 RAID. We use opportunistic reconstruction and constant utilization.

Classic NDS: Write Quorum $w > 2$

The NDS scheme performs operations as described in Section 11.2.4. The average number of tracks in a reliability group is still $\sigma = \frac{s(n-2)+(m-s)(n-1)}{m}$. In contrast to ACATS, the reliability group is fixed and much of the load changes (all, but the operations at the spare) are distributed over the $n-1$ disks in the reliability group instead over the $(n-1)m$ disks in different strings. Consequentially, we have for the loads at the disks in the same reliability group:

$$\begin{aligned}
\lambda_r^{df} &= \lambda_r^{old} + \frac{\pi(t)}{nm-1}\lambda_r^{old} \\
\lambda_w^{df} &= \lambda_w^{old} - \frac{\rho_w}{\rho_r + w\rho_w}\frac{(w-1)}{(n-1)}\zeta + \frac{\pi(t)}{nm-1}\lambda_w^{old} \\
\lambda_t^{df} &= \frac{\sigma+2-w}{(n-1)}\phi + \frac{\rho_r}{\rho_r + w\rho_w}\cdot\frac{\sigma+2-w}{(n-1)}\zeta \\
&\quad + \frac{\rho_w}{\rho_r + w\rho_w}\cdot\frac{\sigma}{(n-1)}\zeta + \frac{(w-1)\rho_w}{\rho_r + w\rho_w}\cdot\frac{\sigma-2w+3}{(n-1)}\zeta \\
&\quad + \frac{\phi+\zeta}{nm-1}
\end{aligned}$$

The behavior after a disk failure is very similar to that in the scheme with reconstruction of spares.

Classic NDS: Example (continued)

We continue our example. In Figure 11.27 we show the utilization at the classic One Dimensional RAID and in Figure 11.28 for the classic MDS based RAID. As one can see, the peak utilization is quite high and the reconstruction time relatively long.

11.3.3 Safe Distributed Sparing

The derivation of the load formulae for SDS mimics the one in Section 11.3.2. We content ourselves with giving the formulae:

Classic SDS: Write Quorum $w = 2$

If the write quorum is two, we have:

$$\begin{aligned}
\lambda_r^{df} &= \lambda_r^{old} + \frac{\pi(t)}{nm-1}\lambda_r^{old} \\
\lambda_w^{df} &= \lambda_w^{old} - \frac{1-\pi(t)}{(n-1)}\lambda_w^{old} + \frac{\pi(t)}{nm-1}\lambda_w^{old}
\end{aligned}$$

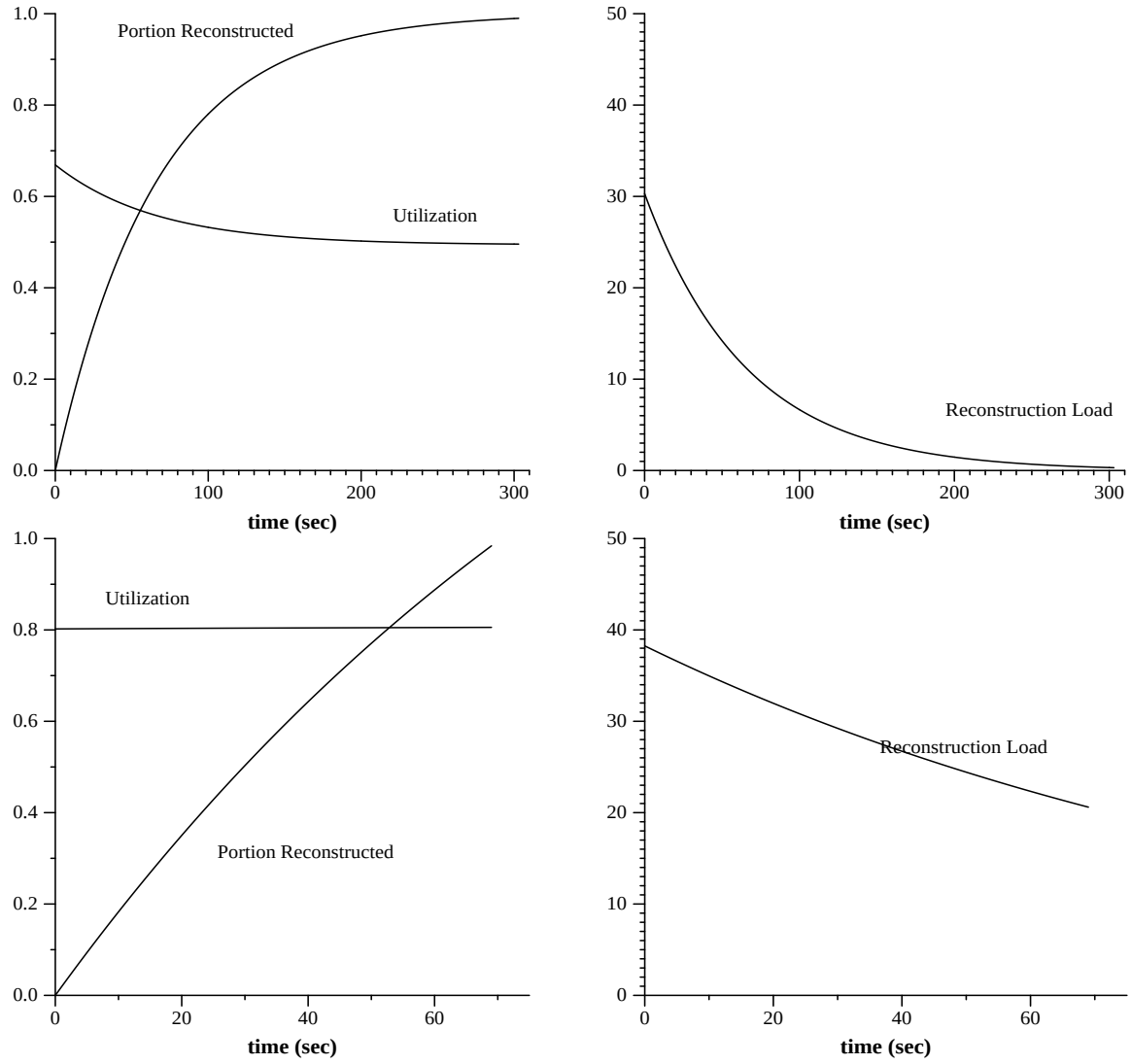


Figure 11.28: Utilization at a Classic MDS RAID using NDS. We show opportunistic reconstruction only and constant utilization.

$$\begin{aligned}
\lambda_t^{df} &= \frac{\sigma}{(n-1)}(\phi+\zeta)+\frac{1}{nm-1}(\phi+\zeta) \\
\lambda_{tw}^{df} &= \frac{1}{nm-1}\frac{m-1}{m}(\phi+\zeta) \\
\zeta &= (1-\pi(t))\lambda^{old}.
\end{aligned}$$

Classic SDS: Write Quorum $w > 2$

If the write quorum exceeds two, we have:

$$\begin{aligned}
\lambda_r^{df} &= \lambda_r^{old}+\frac{\pi(t)}{nm-1}\lambda_r^{old} \\
\lambda_w^{df} &= \lambda_w^{old}-\frac{\rho_w}{\rho_r+w\rho_w}\frac{(w-1)}{(n-1)}\zeta+\frac{\pi(t)}{nm-1}\lambda_w^{old} \\
\lambda_t^{df} &= \frac{sigma+2-w}{(n-1)}\phi+\frac{\rho_r}{\rho_r+w\rho_w}\cdot\frac{\sigma+2-w}{(n-1)}\zeta \\
&\quad +\frac{\rho_w}{\rho_r+w\rho_w}\frac{\sigma}{(n-1)}\zeta+\frac{(w-1)\rho_w}{\rho_r+w\rho_w}\frac{\sigma+(3-2w)}{(n-1)}\zeta+\frac{\pi(t)}{nm-1}(\phi+\zeta) \\
\lambda_{tw}^{df} &= \frac{1}{nm-1}\frac{m-1}{m}(\phi+\zeta)
\end{aligned}$$

We present the disk utilization under SDS in Figures 11.29 and 11.30.

11.3.4 Distributed Sparing of a String

In both the disk failure and the string failure case, the RAID will develop peak utilization equal to that of a RAID with same dimensions with ACATS for the string failure case. We can decrease the load in the disk failure case by distributing the virtual spare disks over all disks in the RAID, instead over all disks in the appropriate reliability group. Then the behavior is that of the NDS scheme.

11.4 The Two-Dimensional RAID

We only consider the two dimensional RAID organization with ACATS. Without ACATS, a two dimensional RAID shows the same poor performance as a Level 4 RAID. There is little performance difference between an MDS based RAID with 2 check disks per reliability group and ACATS and the two-dimensional RAID.

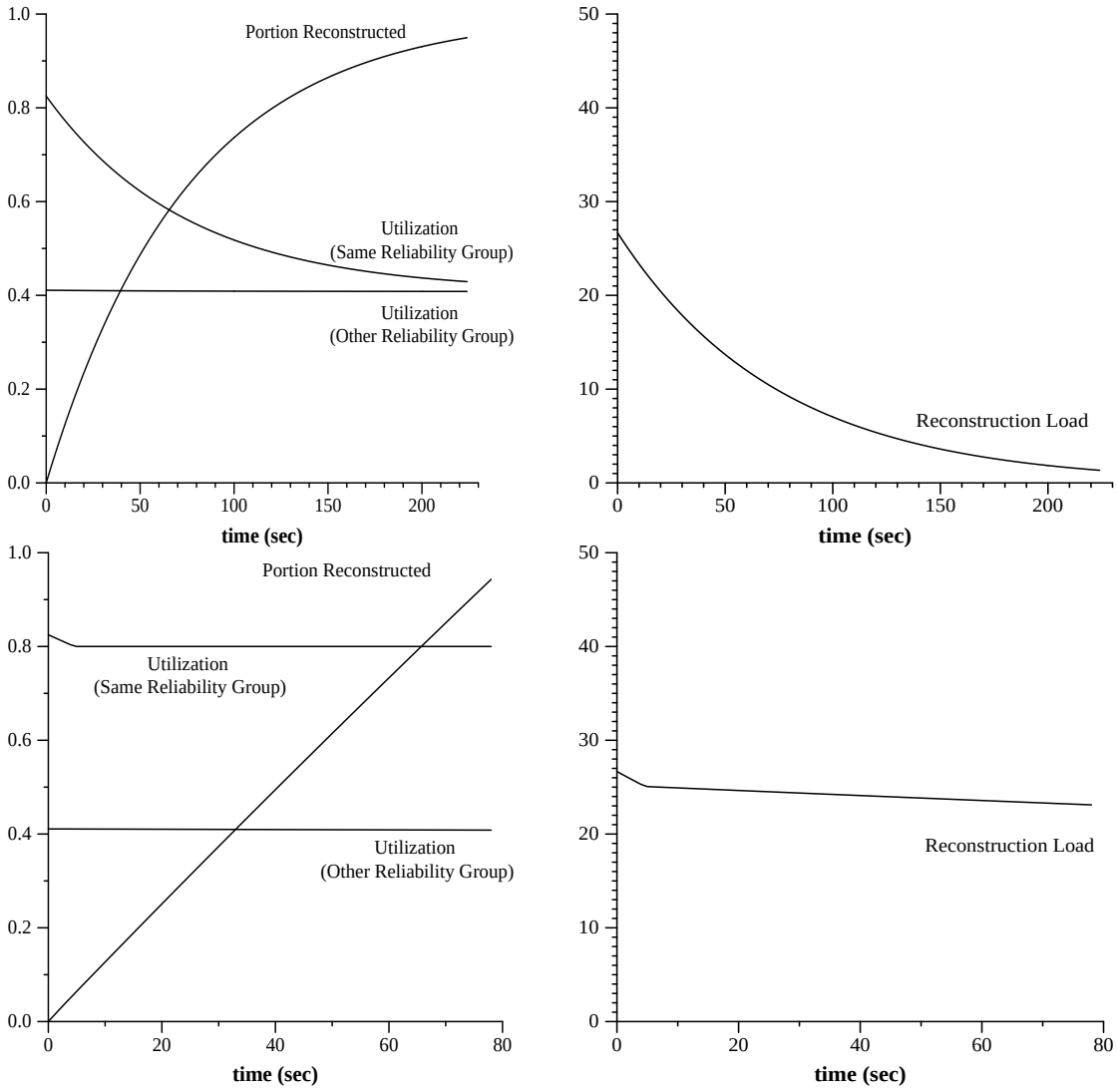


Figure 11.29: Effects of SDS on the Classic One-Dimensional RAID: Opportunistic Reconstruction and Constant Peak Utilization.

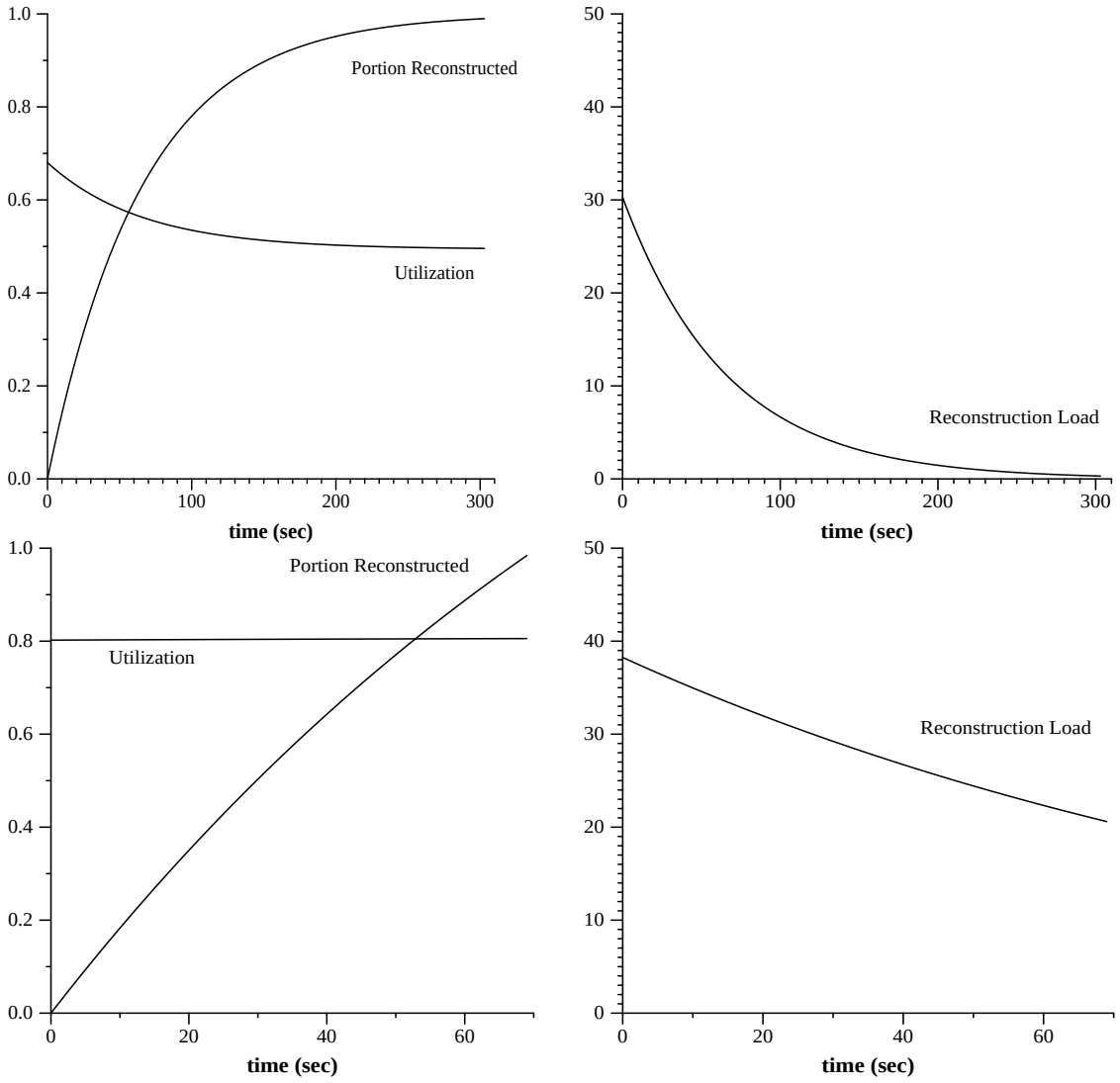


Figure 11.30: Effects of SDS on the Classic MDS based RAID: Opportunistic Reconstruction and Constant Peak Utilization.

We assume that the RAID consists of $N = n^2 + 2n$ disks and is organized in s strings of d disks each. If a disk failed, then there are $N - d$ disks not in the same string. ACATS will place these disks with the same probability of $\frac{2r-1}{N-d}$ in a same reliability group as the failed disk.

11.4.1 No Spares

Two Dimensional RAIDs: Disk Failure:

In case of a disk failure, a write to the failed disk as a check disk proceeds as a normal write involving one less disk. As long as we are not using strong write synchronization, we do not see a change in load or utilization at the other disks.

If we write to the failed disk as a message disk, we need to read the disks of one reliability group (n reads), calculate the old data on the failed disk, then the Δ value. Then we can write the check data in the other reliability group and do a simple write (which counts as a read) to the check disk in the reliability group just read. The load of writes to the failed disk as a message disk is $\rho_w \Lambda / N$. Our analysis is slightly on the pessimistic side, since it stands to reason that for normal loads the second access to one of the check disks does not have a second seek time component. The write load is slightly reduced at all other disks.

A read to the failed disk involves reading all the other n disks in one of the reliability groups.

We obtain for the disks outside the string with the failed disk:

$$\begin{aligned}
\lambda_r^{df} &= \frac{rho_r \Lambda}{N} + \frac{(n+1)\rho_w \Lambda}{N(N-d)} + \frac{n\rho_r \Lambda}{N(N-d)} \\
&= \left(1 + \frac{n}{N-d}\right) \cdot \lambda_r^{old} + \frac{n+1}{3(N-d)} \lambda_w^{old} \\
\lambda_w^{df} &= \frac{3\rho_w \Lambda}{N} - \frac{\rho_w \Lambda}{(N-d)N} \\
&= \left(1 - \frac{1}{3(N-d)}\right) \lambda_w^{old}
\end{aligned}$$

No Spares: String Failure

The disks in a string all belong to different reliability groups. Hence, we need to multiply

the load changes with the number of disks in the failed string to obtain the new load:

$$\begin{aligned} \lambda_r^{sf} &= \left(1 + \frac{dn)}{N-d}\right) \cdot \lambda_r^{old} + \frac{d(n+1))}{3(N-d)} \lambda_w^{old} \\ \lambda_w^{df} &= \left(1 - \frac{d}{3(N-d)}\right) \lambda_w^{old} \end{aligned}$$

No Spares: Example

We choose the middle sized one our two dimensional RAIDs in Chapter 5 as an example. This RAID contained 16 strings with 5 disks each. A reliability group consisted of 8 disks. We again use a read to write ratio of 2:1. The utilization of disks is then 0.33333Λ. After a disk failure, the figure increases to 0.34488Λ or an increase of 4.667% . The utilization in the string failure case reaches 0.38611Λ for an increase of 15.83% . The impressive numbers are explained by the worse capacity to number of disks ratio and by the small number of disks in a string.

11.4.2 Reconfiguration on one Dedicated Check

Our scheme is a simple translation of the Reconfiguration on Check schemes considered previously. Whenever we access data stored as message on the failed disk, we reconstruct the track on one of the check disks. After repair, the RAID has to reconstruct the original distribution of check and message data. Because this can be done at a slow rate, we are only interested in the peak utilization after a failure.

We perform a read from the failed disk by reading the $n-1$ tracks in one of the reliability groups, and reading and the writing the check disk in the reliability group. We perform a write to the failed disk (as message disk) by accessing all the message tracks in one of the reliability groups and then do a track write (involving a previous track read) at the check disk in the reliability group. At the other check disk, we do a normal write, because we have the Δ value available. The response times of these two operations are relatively poor.

The probability π that data is already stored on the check disk increases quickly. The disks outside the string with the failed disk will share the read load originally directed to the failed disk. In addition, the write load in the strings not containing the failed disk will suffer a permanent change. One reliability group will not be having a check disk. Imagine

that this reliability group is the first column. Any write addressed to this column will only generate one instead of two check writes. Hence, the total write load will decrease from $3\rho_w\Lambda$ to

$$\frac{n-1}{n}3\rho_w\Lambda + \frac{1}{n}2\rho_w\Lambda.$$

This decrease is however only experienced by the $N-d$ disks outside the string with the failed disk. Hence, their write load decreases by $\frac{1}{N-d}\frac{1}{n}\rho_w\Lambda$ which, in terms of the old write load, is $\frac{N}{3(N-d)n}\lambda_w^{old}$. Because in addition these disks take over the write load of the failed disk, we see an increase in the write load of $\frac{\pi}{(N-d)}2\rho_w\Lambda$. As we have remarked, one write results from attempted writes to the failed disk in its capacity of message disk. This increases the write load by $\rho_w\zeta$ (the load of this kind) divided by $N-d$.

$$\begin{aligned}
\lambda_r^{df} &= (1+\frac{\pi}{(N-d)})\lambda_r^{old} \\
\lambda_w^{df} &= \lambda_w^o ld - \frac{\rho_w\Lambda}{n(N-d)} + \frac{\pi 2\rho_w\Lambda}{N-d} + \frac{\rho_w\zeta}{N-d} \\
\lambda_t^{df} &= (n-1)(\phi+\zeta) \\
\lambda_{tw}^{df} &= (\phi+\zeta) \\
\zeta &= (1-\pi)\frac{\Lambda}{N}
\end{aligned}$$

For a string failure, we observe the differences magnified d times:

$$\begin{aligned}
\lambda_r^{ds} &= (1+\frac{d\pi}{(N-d)})\lambda_r^{old} \\
\lambda_w^{ds} &= \lambda_w^o ld - \frac{d\rho_w\Lambda}{n(N-d)} + \frac{d\pi 2\rho_w\Lambda}{N-d} + \frac{\rho_w\zeta}{N-d} \\
\lambda_t^{df} &= (n-1)(\phi+\zeta) \\
\lambda_{tw}^{df} &= (\phi+\zeta) \\
\zeta &= (1-\pi)\frac{d\Lambda}{N}
\end{aligned}$$

11.4.3 Use of a Spare Disk

After data is reconstructed on the spare disk, the utilization at all other disks is the same as before.

For a read to the failed disk we read n disks in the same reliability group. If we write to the failed disk as a message disk, we read a track at the $n-1$ other message disks in one of the reliability groups to which the failed disk belongs, then, we read the track at the check disk in this reliability group and write the new block; this counts as a write operation. With the information gathered, we can now reconstruct a track's worth on the spare disk and also use the Δ value to write to the other check disk in a normal way. We handle a write that uses the failed disk as a check disk by normally writing to the message and the other check disk. To reconstruct the track data on the failed disk, we read the other $n-1$ disks in the same reliability group, but we do not need to read the track at the message disk, because we already swept through the whole track during the write and just need to store this information.

Hence, for the disks outside the string with the failed disk, (as well as for the disks on this string,) the read and the write load proper has not changed. In addition, depending on the character of the reconstruction, we read n tracks (for a forced reconstruction and an opportunistic reconstruction request arising from a read request) or $n-1$ tracks (for a write triggered opportunistic reconstruction.)

We obtain for the disks outside the string with the failed disks:

$$\begin{aligned}
\lambda_r^{df} &= \lambda_r^{old} \\
\lambda_w^{df} &= \lambda_w^{old} \\
\lambda_t^{df} &= \frac{n}{N-d}\left(\frac{\rho_r}{\rho_r+3\rho_w}\zeta+\phi\right)+\frac{n-1}{N-d}\frac{3\rho_w}{\rho_r+3\rho_w}\zeta \\
\zeta &= (1-\pi)(\lambda_r^{old}+\lambda_w^{old})
\end{aligned}$$

At the spare, the load is

$$\begin{aligned}
\lambda_r &= \pi\lambda_r^{old} \\
\lambda_w &= \pi\lambda_w^{old} \\
\lambda_t &= \phi+\zeta
\end{aligned}$$

If we compare with a similarly dimensioned MDS based RAID with 2 check disks per reliability group, we see that the utilization is only slightly higher for the two dimensional scheme.

Reconstruction on Check: Example (continued)

Using opportunistic reconstruction, we observe an initial utilization increase of 9.14%, that quickly decreases to the original utilization. We present the results in Figure 11.31

11.4.4 Naive Distributed Sparing

In naive distributed sparing, the load at the spare is equally distributed over the remaining $N-1$ disks. We obtain for the loads at the disks outside the string with the failed disk:

$$\begin{aligned}
\lambda_r^{df} &= \left(1+\frac{\pi}{N-1}\right)\lambda_r^{old} \\
\lambda_w^{df} &= \left(1+\frac{\pi}{N-1}\right)\lambda_w^{old} \\
\lambda_t^{df} &= \frac{n}{N-d}\left(\frac{\rho_r}{\rho_r+3\rho_w}\zeta+\phi\right) \\
&\quad +\frac{n-1}{N-d}\frac{3\rho_w}{\rho_r+3\rho_w}\zeta+\frac{\phi+\zeta}{N-1} \\
\zeta &= (1-\pi)(\lambda_r^{old}+\lambda_w^{old})
\end{aligned}$$

NDS: Example (continued)

We extend our example to the NDS scheme. The utilization (see Figure 11.32) increases initially to about 10% and then decreases to about 1.7% .

11.4.5 Safe Distributed Sparing

We write the reconstructed data in safe distributed sparing using a track write operation, involving either two or three disks, depending on whether the spare space is located in a same reliability group or not.

First, we need to calculate the probability that an arbitrary disk is in a same reliability group than an arbitrarily chosen one. A given disk serves for a random track as a check disk with probability of $2n/N$ and as a message disk with probability n^2/N. A check disk has n other disks in the same reliability group and a message disk has $2n$. The probability of another random disk to share a reliability group is hence

$$p_2 = \frac{2(n^2+n^3)}{N(N-1)}$$

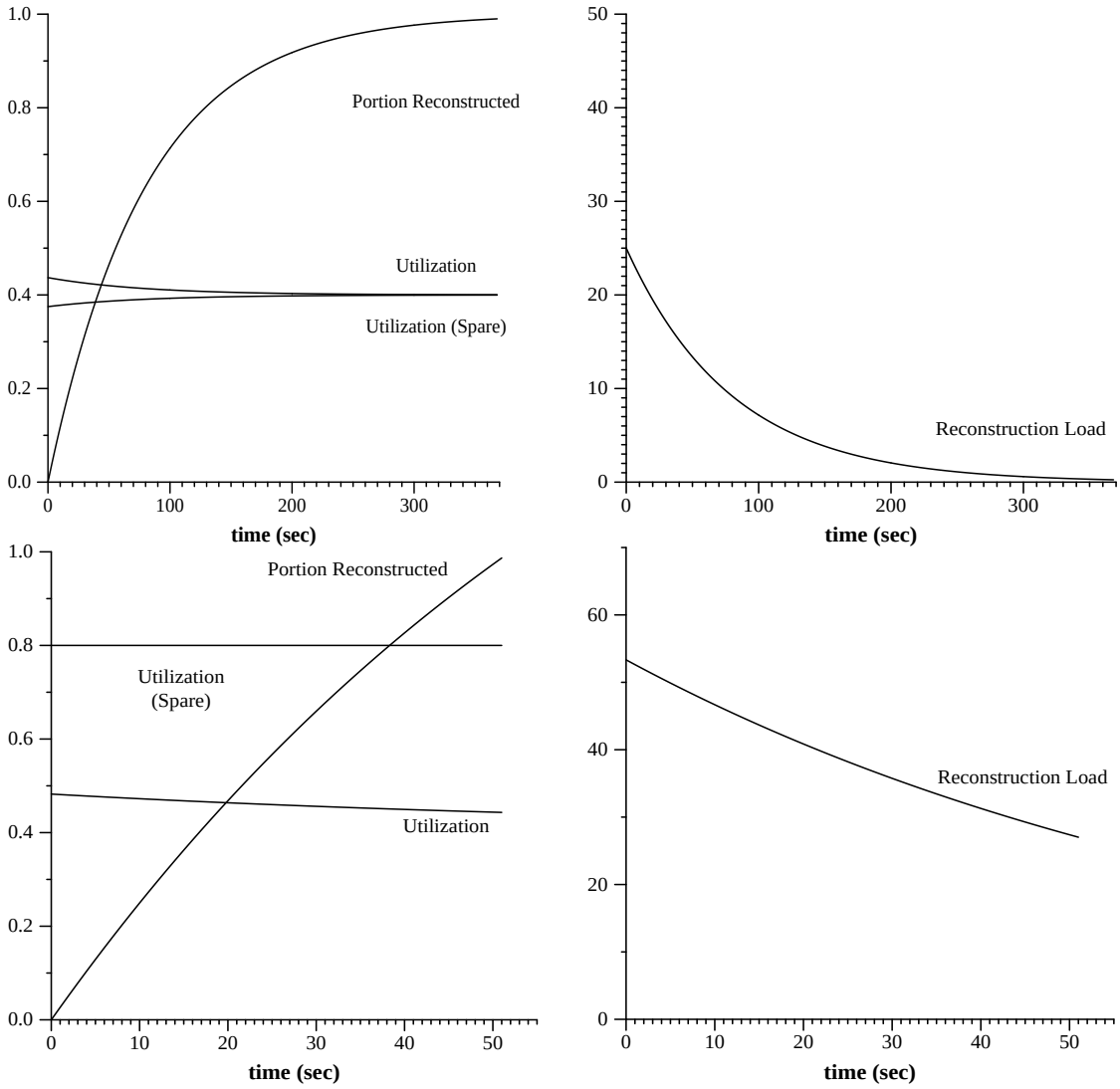


Figure 11.31: Utilization at the Two Dimensional RAID after a Disk Failure using Reconstruction on Spare Disk. We use opportunistic reconstruction (top) and constant spare disk utilization (bottom).

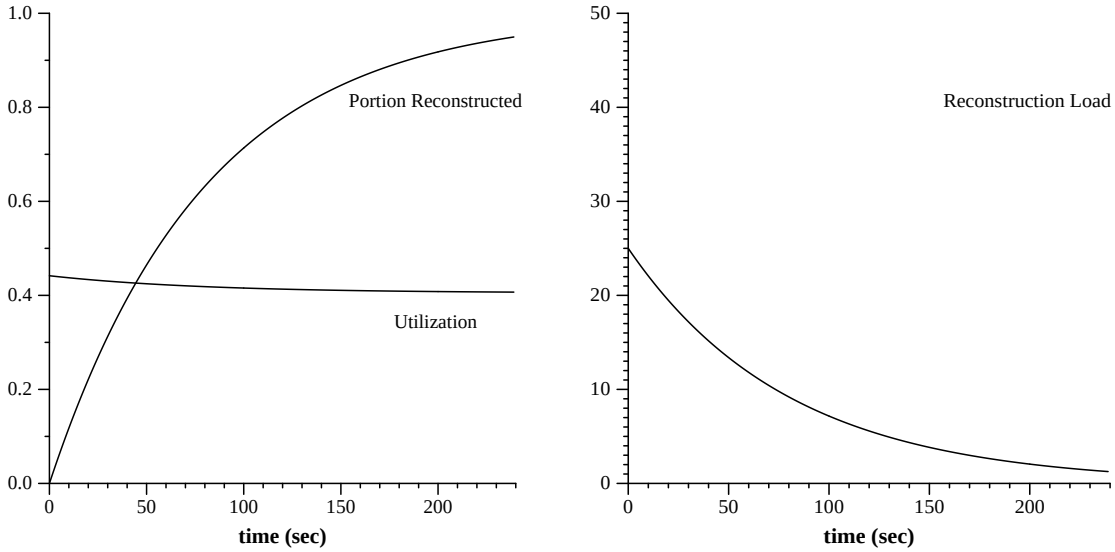


Figure 11.32: Utilization at the Two Dimensional RAID after a Disk Failure under NDS with Opportunistic Reconstruction

which coincides with the probability that a reconstruction operation needs only two instead of three write operations. The expected number of track writes per reconstruction on spare space is then (with $p_3 = 1 - p_2$)

$$(2p_2 + 3p_3)(\phi + \zeta)$$

We can calculate the load at the disks outside the string with the failed disk by adjusting the load at the spare accordingly and then distribute the load over the right set of disks.

$$\begin{aligned}
\lambda_r^{df} &= \left(1 + \frac{\pi}{N-1}\right)\lambda_r^{old} \\
\lambda_w^{df} &= \left(1 + \frac{\pi}{N-1}\right)\lambda_w^{old} \\
\lambda_t^{df} &= \frac{n}{N-d}\left(\frac{\rho_r}{\rho_r + 3\rho_w}\zeta + \phi\right) + \frac{n-1}{N-d}\frac{3\rho_w}{\rho_r + 3\rho_w}\zeta \\
\lambda_t w &= \frac{2p_2 + 3p_3}{N-1}(\phi + \zeta) \\
\zeta &= (1-\pi)(\lambda_r^{old} + \lambda_w^{old})
\end{aligned}$$

We can, again, observe that the load is slightly higher than for a similarly dimensioned MDS RAID.

SDS: Example (continued)

We present the utilization behavior under SDS in Figure 11.33.

11.4.6 Distributed Sparing of a String

The effects of a disk failure are the same as in NDS. We treat the case of string failure only. The analysis parallels that for the use of a spare string and its adjustment to naive distributive sparing.

$$\begin{aligned}
\lambda_r^{ds} &= \left(1 + \pi d \lambda_r^{old}\right) \lambda_r^{old} \\
\lambda_w^{ds} &= \left(1 + \pi d \lambda_r^{old}\right) \lambda_w^{old} \\
\lambda_t^{ds} &= d\frac{n}{N-d}\left(\frac{\rho_r}{\rho_r + 3\rho_w}\zeta + \phi\right) + d\frac{n-1}{N-d}\frac{3\rho_w}{\rho_r + 3\rho_w}\zeta + \frac{(\phi + \zeta)}{N-d} \\
\zeta &= (1-\pi)d(\lambda_r^{old} + \lambda_w^{old})
\end{aligned}$$

Distributed String Sparing: Example (continued)

We present the utilization behavior after a string failure in Figure 11.34. The large number of strings implies that more disks survive a string failure and share the reconstruction work equally compared to the MDS organization. The peak disk utilization is only 52.5% over baseline load as a consequence of the higher hardware costs.

11.5 A Metric for Reconstruction Speed

A storage unit failure (disk or string) leads to a utilization peak for most disks in the RAID. The speed of reconstruction on spare measures the inconvenience users might experience during this period. The reconstruction time also influences the reliablity of the RAID. However, as we have seen, reconstruction times of less than a minute are hardly noticeable. ACATS and other declustering techniques that spread the workload over a larger number of disks reduce reconstruction speeds in the disk failure case considerably. This positive effect on reliability is far outweighed by the larger probability that certain unit failures lead to data loss and the MTTDL values are lower.

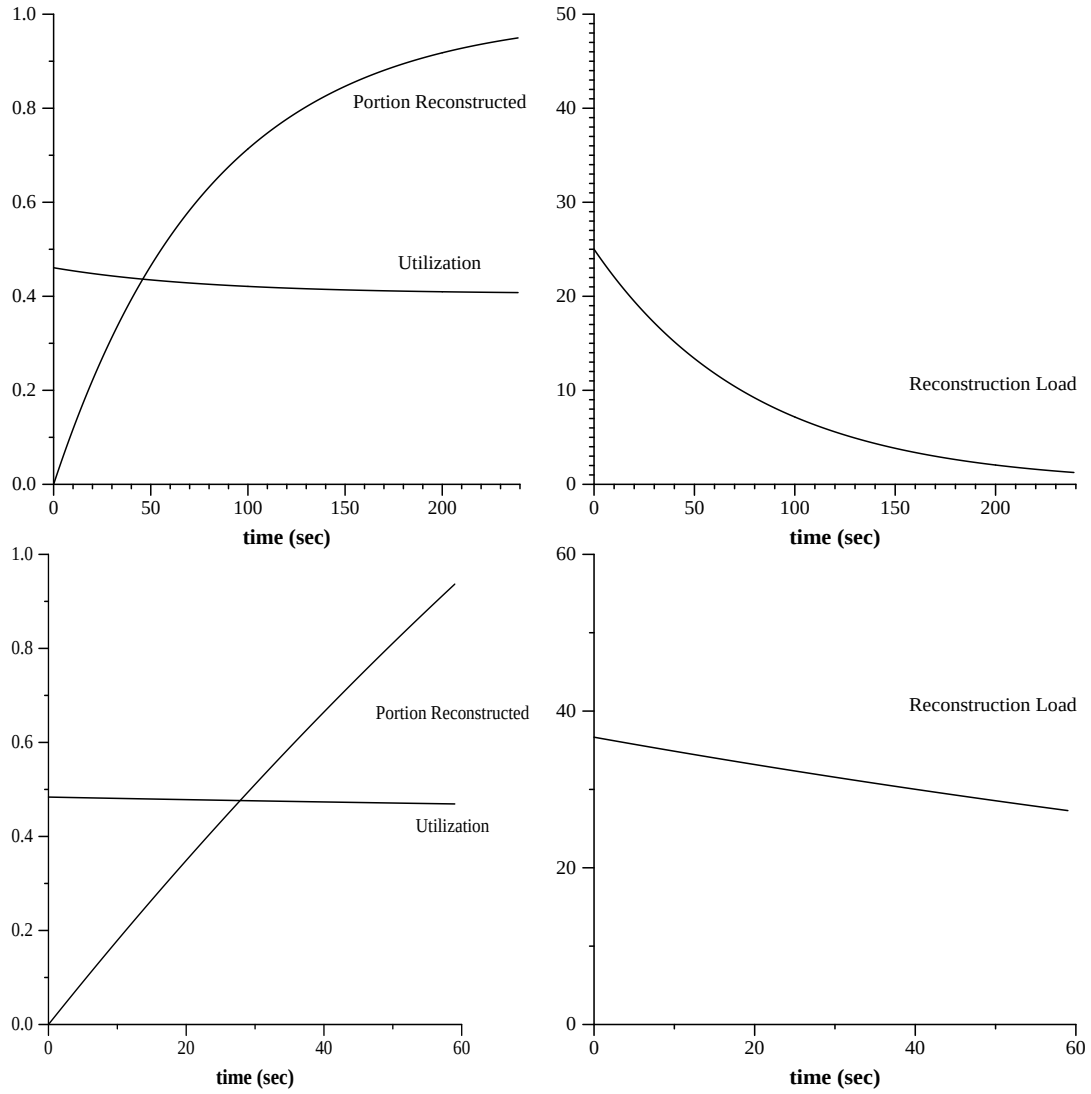


Figure 11.33: Utilization at the Two Dimensional RAID after a Disk Failure under SDS. We use opportunistic reconstruction (left) and constant disk utilization (right). (Reconstruction load is given in tens of requests per millisecond.)

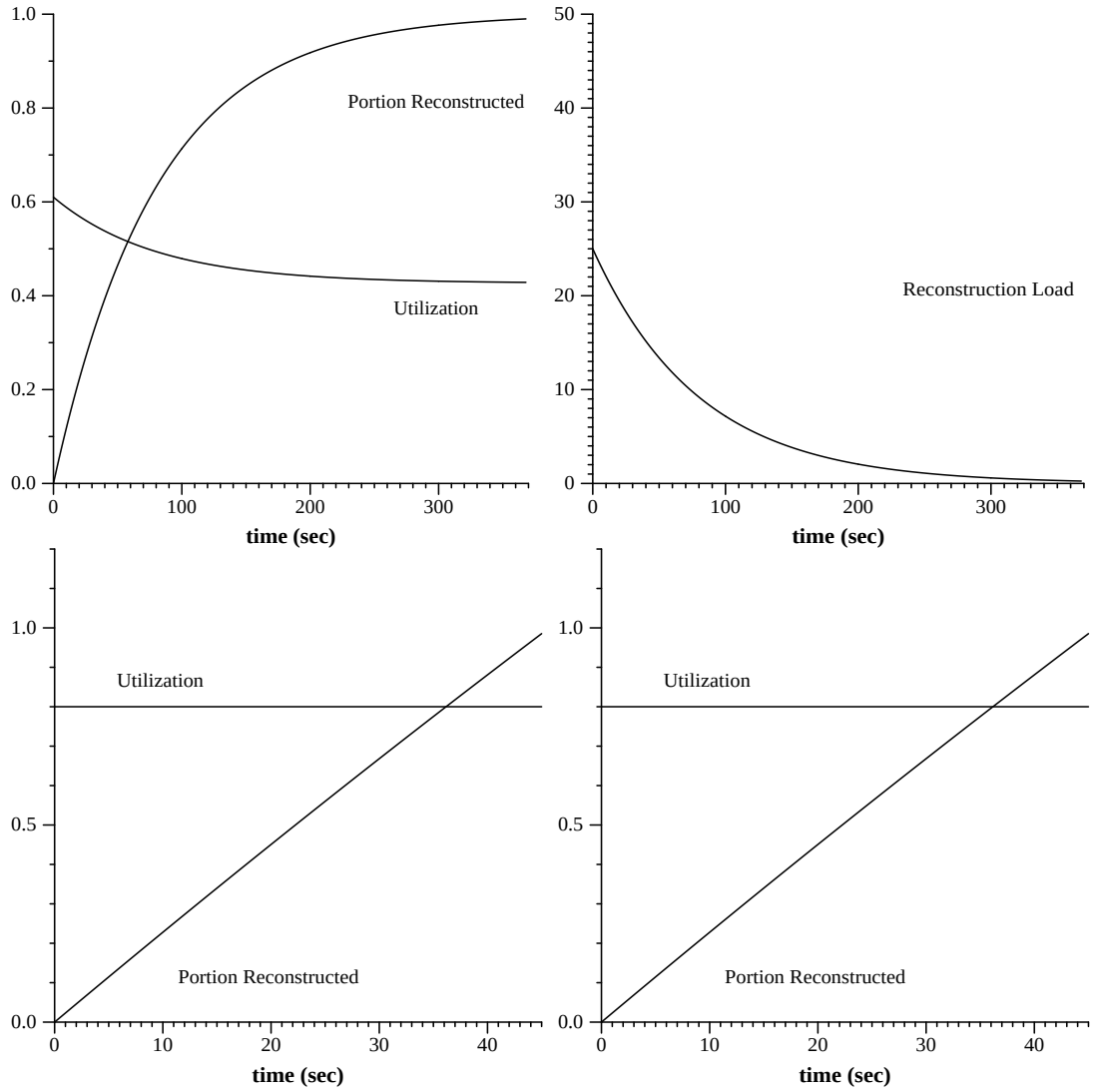


Figure 11.34: Utilization at the Two Dimensional RAID after a Disk Failure under SDS. We use opportunistic reconstruction (left) and constant disk utilization (right). (Reconstruction load is given in tens of requests per millisecond.)

RAID Organization	Reconstruction Time (sec)
Level 5 ACATS: NDS	22.45
Level 5 ACATS: SDS	23.65
Level 5 ACATS: String DS	20.19
MDS ACATS: NDS	20.19
MDS ACATS: SDS	21.26
Level 5: Classic NDS	112.26
Level 5: Classic SDS	118.25
Level 5: Classic String DS	100.94
MDS Classic: NDS	100.94
MDS Classic: SDS	106.32

Table 11.2: Reconstruction Times for RAID Organizations with NDS, SDS and String Distributed Sparing

In order to capture reconstruction speed in a single metric (akin to the MIPS computer speed ratings) we make several assumptions. The resulting metric values are not necessarily meanigful as absolute numbers but capture well the relative benefits of the schemes.

Our assumptions are:

1. The baseline utilization at each disk is 30%. This is at the high end of current RAID utilization. At this utilization the response response time to a simple read is about 1.43 times the service time.

2. The reconstruction on spare uses both optimistic and forced reconstruction and targets a utilization at the disks of 60%. At this rate the response time is about 2.50 times the service time.

3. The probability of a *hit* (i.e. an access finds an item already reconstructed) is the proportion κ of already reconstructed data on spare. We make the modeling choice because there are no reasonable assumptions about the probability and because the assumption leads to an easily solvable ordinary equation. Our assumption assumes no locality at all.

The baseline utilization at a disk is u_0. The target peak utilization at a disk during the reconstruction is u_c. The failed disk (or string) is replaced by spare space distributed

evenly over M disks. The reconstruction workload W expresses in seconds the effort to reconstruct a single track. It is evenly distributed over N disks. Finally, we denote opportunistic and forced reconstrustion by ψ given in requests per second. We express the target utilization u_c during reconstruction as baseline utilization, added work from the failed disk and reconstruction work. This target utilization is valid for a disk that contains both spare space used to replace storage space on the failed disk and that is part of the reconstruction effort. The formula does e.g. not apply to disks on the same string, because they never take part in the reconstruction effort:

$$u_c = u_0 + \kappa \cdot \frac{u_0}{M} + \psi \cdot \frac{W}{N}.$$

As

$$\kappa(t) = T^{-1} \int_0^t \psi(\tau) d\tau$$

with disk capacity T, we obtain through differentiation

$$\frac{u_0}{TM} \cdot \psi + \frac{W}{N} \cdot \psi'$$

which is solved by

$$\psi = C \cdot \ln(-Rt); \qquad \text{with} \quad R = \frac{u_0 N}{TMW}$$

By substitution in the original equation we determine

$$\begin{aligned} \psi &= \frac{N(u_c - u_0)}{W} \cdot \exp(-Rt) \\ \kappa &= \frac{(u_c - u_0)M}{u_0} \cdot (1 - \exp(-Rt)) \end{aligned}$$

By setting $\kappa = 1$ and using $u_c - u_0 = u_0$ we obtain

$$t_{rec} = -R^{-1} \ln(1 - M^{-}1)$$

for the reconstruction time. An approximation is

$$t_{rec} = \frac{TW}{u_0 N}.$$

We give the reconstruction times for our examples in Table 11.2. The two-dimensional RAID has the same behoviour as the MDS RAID with ACATS in our slightly simplified model. The times quoted are for reconstruction after disk failure. For string failure, the times for the classic organizations are unchanged and the times for the ACATS based organizations are identical to the times for the classic sister organization. We can conclude that the change from NDS to SDS lengthens the reconstruction process by about 5%.

11.6 Synthesis

We have measured performance in terms of disk utilization. Peak disk utilization determines performance for all classes of requests through the time for a device to clear.

In all instances, ACATS has proven its superior performance in disk failure scenarios, just as Muntz and Liu ([25]) have predicted. Reconfiguration on Check makes the best of a bad situation, if no spare space is available, by quickly gaining the performance benefits of unrelated disks. By updating whole tracks we gain fast reconstruction and reconfiguration times.

Comparison of the load formulae show that the two-dimensional RAID preforms almost as good as a MDS RAID with 2 check strings in failure cases.

It seems that we need to store large amounts of data for reconstruction processes, which would put a severe strain on the non-volatile cache. However, the properties of the parity code and MDS codes in general imply that we do not have to wait for all data fragments to arrive from different disks; instead, we only need to keep one intermediate result to which the fragments can be added.

Chapter 12

Application of MDS Codes for Data Bases

We have introduced the use of MDS codes for RAIDs. In this chapter we apply the same storage idea to Distributed Databases. Academic research in this field has centered mainly on the development of secure update protocols which has to synchronize among sites in a chaotic environment. Our contribution here presupposes such a scheme and concentrates on the storage scheme and the conditions it poses to the protocols.

Data bases are distributed over several sites to allow users faster and more reliable access. Multiplying storage needs and much more complicated update protocols are the price to pay. Our scheme trades the advantage of reading local data for much lower storage costs. It uses MDS code data dispersion to make data very available without keeping copies.

12.1 Storage Scheme

The database is organized in equal sized *pages.* The pages are the atomic storage units: each data access addresses a whole page. The pages themselves are organized in blocks of n pages each. We treat the bytes in parallel position as the information symbol of a linear, systematic MDS code and store the check bytes in m *check or secondary pages.* A block consists then of n *primary* pages and m secondary pages. The MDS property of the code used assures us, that the database is available for reads if only n sites are accessible.

Our basic scheme stores the $m + n$ pages of a block at as many sites. All data is

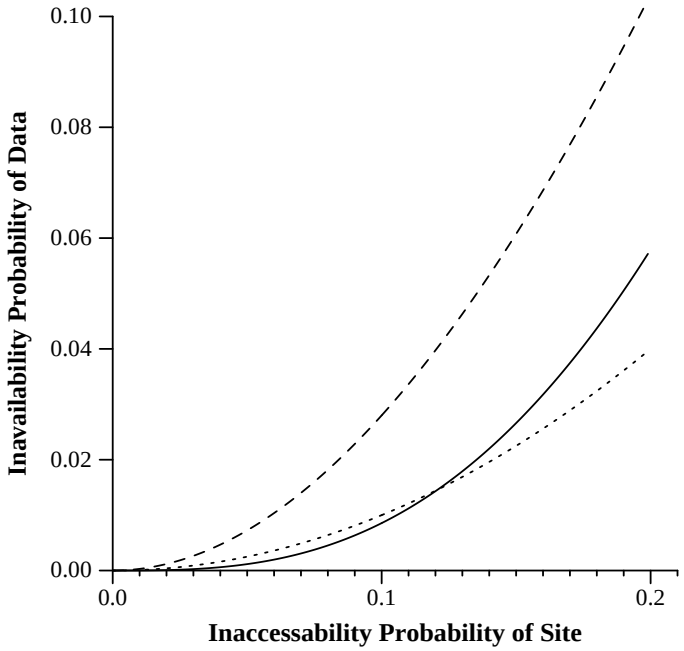


Figure 12.1: Data inavailability in dependence on site inavailability probability for the MDS storage scheme with 3 primary and 2 secondary pages per block (solid line), a triple (dotted) and a double replicated database.

available when at least m sites can be accessed. As m can be smaller than n, our scheme achieves excellent data availability for a small cost in storage. In Figure 12.1 we give the data availability for our scheme and compare it with two other schemes. Our scheme uses 3 primary pages and 2 secondary pages per block and we compare it with a double and a triple replicated distributed data base. The higher number of sites makes our scheme safer while the total storage amount, 1.67 times the size of all records, is smaller than the storage needs of the other schemes which is 2 or 3 times this amount, respectively. We can even prove mathematically that a "storage optimal" database system has to use an MDS code.

The storage efficiency of our scheme comes at a price. If a single site becomes inavailable, then access to the primary pages stored there have to be reconstructed by accessing m other sits. In a read oriented environment this can almost double the workload if a site has failed. To balance the site workload we distribute the primary and secondary pages of a block over all the sites. The lay-out is illustrated in Figure 12.2 (upper half)

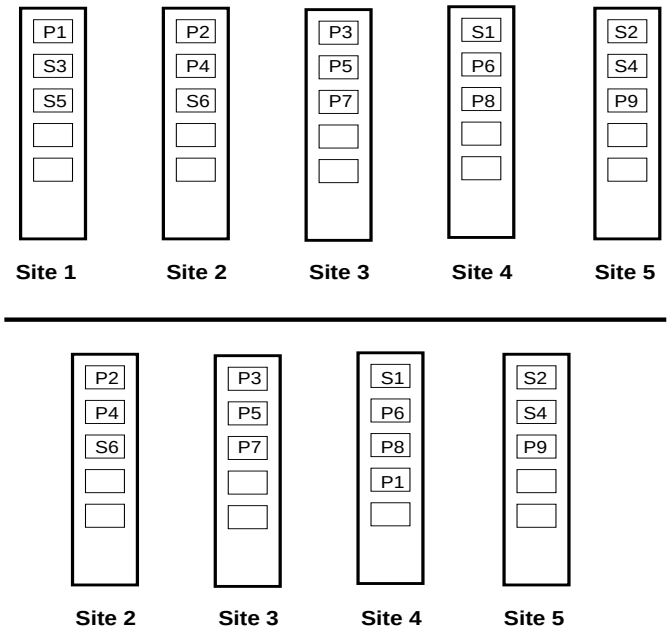


Figure 12.2: The Storage Scheme with $m = 3$ and $n = 5$ before and after Loss of a Site.

where the first block consists of primary pages P1, P2, P3 and secondary pages S1, S2, the second block of primary pages P4, P5, P6 and secondary pages S3 and S4. The workload is now evenly distributed before and after a site becomes inaccessible. In some instances, access to data shows temporal locality, e.g., if the scheme is used to store archival data. To further reduce the workload after a presumed site failure, we can cache reconstructed data as is shown in Figure 12.2 (lower half). After Site 1 has become inaccessible, an access to page P1 results in accesses to P2, S1 and S2 to reconstruct page P1. The reconstructed page is then stored for further use at Site 4. The replacement sites are chosen by a scheme that is transparent to all sites.

12.2 Read and Write Protocol

We assume that the database runs a local serializability protocol at each site. Furthermore, we assume a global consistency protocol, e.g., one based on votes, to assure consistency among the database sites.

The basic write operation is in principle the same as for the updates in an MDS

RAID. A write that changes a primary page needs to propagate the Δ value between the old and the new page to each site containing a secondary site. There, the new secondary page is calculated from this Δ value and the old secondary page.

A typical update scheme would first read the old records and then propagate the the Δ value to all sites. The other primary sites do not need to change their local storage contents but only act as voters.

A typical read would gather enough votes to achieve a read quorum, and preferably read from the site which stores the primary page that contains the record requested. Alternatively, n pages of the same superblock are gathered and the primary page is reconstructed from these.

12.3 Performance

We assume a write and read quorum of q. In the absence of an inaccessible site and , a small read request involves $2(n + m)$ messages and one transmission containing the primary site if the primary page is not stored locally, which happens with probability $1/(n + m)$. A write can be organized with the same number of messages and transmission of only $1 + m$ pages. If there is an recognized inaccessible site, the expected number of page transmissions for a read increases from $(n + m - 1)/(n + m)$ to $2n + m - 2/(n + m)$ because a primary page stored at the inaccessible site needs to be reconstructed from n pages, of which one is locally available.

The need for data transmission is the major drawback of our storage scheme and makes the scheme unusable for databases with frequent queries that involve scanning of many records.

12.4 Use for Archival Storage

Our scheme is best adaptable to an archival storage environment in which the most common operation is a record lookup. A more static database management scheme is suited to such an environment. We assume that each site contains enough information to locate records on pages.

To further facilitate the database management, we assume that the assignement

of records to pages is transparent. In addition to the read and write operations, as sketched before, we now introduce a light read operation, which only accesses the site containing the primary page with the record in question. A light update operation is administered by the site containing the primary page and hence avoids the necessity of a read of the page over the network.

To adjust for failed or inaccessible sites, we run a quorum based protocol that excludes and rejoins those sites and which can be initiated by any site that notices a timeout in connection with one site. The consistency control can be facilitated and double checked by a signature scheme.

12.5 Signature Schemes

Signatures are an efficient mean to compare local copies of distributed data bases and thus ensure that discrepancies are quickly detected (see [10],[24].) We discuss signature schemes here because they usually presuppose the actual existence of the data at all sites. We discuss two signature schemes that apply to our storage scheme.

A signature scheme divides the data base contents in pages. It maintains page signatures which compress the data in the page into a small unit of typically a few bytes. While it is possible that different pages have the same signature, the likelihood is only 2^{-l} where l is the length (in bits) of a signature. The page signatures themselves are then compressed into a single data base signature (usually the exclusive OR of all page signatures) which is then used for comparison among sites. If signatures disagree we know that the copies of the data base differ and if they agree we know with the very low error probability 2^{-l} that they agree. Efficient algorithms deal with the diagnostics of a discovered discrepancy.

12.5.1 Linear Schemes

We interpret the bytes of a data base page as the elements of the Galois field with 256 elements and calculate the page signature as one or more bytes obtained as a linear function of all the characters in the page. This method of calculation yields page signatures that can be subjected to the same method of generating checks as the data themselves. A

short calculation reveals that a check of page signatures is the page signature of the check page. Furthermore, to update a check page signature, we only need the Δ value of the updated page signature, which can be directly calculated from the old and the new primary page signature. Consequentially, a site can update signatures for all pages by monitoring the signatures of page updates.

A signature scheme based on linear page numbers behaves in every other aspect as the signature schemes considered in the literature. In the next Section, we present a signature scheme that is radically different.

12.5.2 Smart Version Numbers

Version numbers are a standard tool of consistency control in distributed systems. Our scheme changes version numbers in a history sensitive way using update signatures. Our probabilistic scheme detects lost updates and out of order updates (affecting the same data) with arbitrary high likelihood. It diagnoses any discrepancies between the update histories at different sites.

We assumed that the data base consists of many *pages.* We calculate an update signature from the update command itself, the issuing site and the local time. The update signature is several bytes long and is propagated along with the update command or can be calculated by the receiving sites individually. A site that processes an update calculates both a new page number for the page(s) affected by the update and a general version number, which reflects the history of updates. We think of this version number as a condensed log. The version number is passed along with any write or read requests to other sites where it acts almost as a capability. If the version number of the sending site agrees with the version number of the receiving site then both sites are in agreement: they have processed the same updates in the same order. A small log of recent version numbers limits the number of discrepancy resolution procedures necessary.

Calculation of the Signatures We calculate the update signature with any one of the schemes proposed for signature schemes in general. The calculation of the page signature is more involved. We interpret the update and page signatures as unsigned numbers between 0 and $2^l - 1$, where l is the length in bits. The new page number p_{new} is calculated from

the old page number p_{old} and the update signature u using

$$p_{\text{new}} = 2p_{\text{old}} + u.$$

Finally, the new version number is the exclusive-or of all page signatures.

Discrepancy Detection Capability Signature scheme cannot be absolutely secure, because they compress information. An optimal update signature scheme will yield the same signature for two different updates with only probability 2^{-l} and it is not difficult for a practical scheme to get close to this value. Similarly, page signatures cannot provide absolute security: Any given page signature can be modified by a single update signature to yield another given page signature. The likelihood of such an unintended aliasing is always 2^{-l} and with this probability our scheme detects missing extraneous updates. Our scheme will always detect two updates that are executed directly out of order and it will detect with probability 2^{-l} that an update has been executed with more than one other request delay. We give the proof below. The version number finally always reflects one errant page signature and more with probability $1 - 2^{-l}$. Metzner's algorithm, which is a version of binary search, will find the offending pages. An implementation of our scheme will probably use a small log of recent updates and thus gain the capability to diagnose most discrepancies ad hoc.

Proofs We give the proofs for the effects of out-of-order updates on the page signatures. Assume that the original page signature is p_{old} and that updates with update signatures u and v are performed out of order at two sites.

Let p_{old} denote the old version number and assume that updates with version numbers u and v are performed out of order at two different sites. These sites calculate page numbers $p_1 = u + 2v + 4p_{\text{old}}$ and $p_2 = v + 2u + 4p_{\text{old}}$. The difference amounts to $u - v$ which is zero if u and v coincide. Now assume that update u is performed after updates v_1, $v_2 \ldots v_n$ at Site One and before at Site Two. The page signatures are respectively

$$\begin{aligned} p_1 &= u + 2v_1 + 4v_2 + \ldots + 2^n v_n + 2^{n+1} p_{\text{old}} \\ p_2 &= v_1 + 2v_2 + \ldots + 2^{n-1} v_n + 2^n u + 2^{n+1} p_{\text{old}} \end{aligned}$$

The difference is zero if

$$v_1 + 2v_2 + \ldots 2^{n-1}v_n = (2^n - 1)u$$

Because $2^n - 1$ is odd, the equality holds for exactly one out of 2^l signatures.

A Mutation of the Scheme If there is a danger of applying updates to the wrong pages, a more complicated page signature and version number scheme can be used. In this alteration, the signatures and version numbers are interpreted as non-zero elements of the Galois Field with 2^l elements. The page signature is updated according to the formula

$$p\text{new} = p_{\text{old}}^2 \cdot u.$$

The data base version number is calculated as a linear form of the page signatures. The scheme can distinguish between updates with the same signature to different pages.

Appendix A

Glossary of Terms

Almost Complete Address Translation (ACATS) A scheme, that translates an external data block address in a two stage procedure to a internal data block address. In one stage, all strings are permuted and in the other disks inside a string are permuted by the addressing scheme.

Check Data RAID generated data that gives redundancy.

Check Disk A disk storing check data.

Disk Failure Pattern (DFP) A set of disks in the RAID, whose failure would lead to irrecoverable data loss in the RAID.

Distributed Sparing A scheme in which spare space amounting to a whole string is distributed over the whole RAID. The utilization of disks is smaller, because each disk contains some spare space.

Forced Reconstruction Reconstruction on spare space that is triggered by the disk controller.

Information Data User generated data that is stored in unchanged form on a disk.

Information Disk (Synonymous to Message Disk) A disk that stores message (=information) data.

Main String A string of disks that carry data under normal operating conditions, as opposed to a string of spare disks.

Message Disk (Synonymous to Information Disk) A disk that stores message (=information) data.

Minimal Disk Failure Pattern (MDFP) A disk failure pattern, that does not contain a smaller Disk Failure Pattern.

Naive Distributed Sparing (NDS) A few disks worth of spare disk is distributed over the whole RAID. The advantages of distributed sparing are made available for smaller amounts of spare space.

Opportunistic Reconstruction Use of write piggy-backing and read redirect to reconstruct data on spare.

Reconfiguration on Check A performance boosting scheme that reconstructs message information on check disks to replace a failed disk or string.

Reconstruction on Spare In the case of disk or string failure, the message contents of the failed disk are written on a/the check disk.

Reliability Group A set of data carriers (like disks themselves or disk blocks). The data in a reliability group shares redundancy that makes reconstruction of data possible in the event of a failure.

Save Distributed Sparing (SDS) A reliability improvement of Naive Distributed Sparing, in which reconstruction of lost disk data uses RAID writes to protect reconstructed data against further component failure.

Single Disk Equivalent Mean Time to Dataloss (SDE MTTDL) A RAID reliability measure that gives the MTTF of a disk such that a collection of disks offering the same amount of storage has the same reliability as the RAID.

Spares' String A string of spare disks as opposed to a Main String.

String A set of disks in a disk array that share components essential for the function of the disks.

Bibliography

[1] Khaled A.S. Abedl-Ghaffar and Amr El Abbadi: *An Optimal Strategy for Comparing File Copies,* Technical Report, University of California, Davis and Santa Barbara, 1992

[2] Divyakant Agrawal and Amr El Abbadi: *Storage Efficient Replicated Databases,* IEEE Transactions on Knowledge and Data Engineering, 1990, p.342-352, volume 2,

[3] Azer Bestavros: *IDA-based Redundant Arrays of Independent Disks,* Proceedings of the First International Conference on Parallel and Distributed Information Systems, December 1991, p. 2-9, Miami Beach,

[4] Mario Blaum, Hsieh Hao, Richard L. Mattson and Jai Menon: *A Coding Technique for Recovery against Double Disk Failures in Disk Arrays,* US Patent Docket #SA889-0443 pending, 1989

[5] Walt Burkhard and Jai Menon: *MDS Disk Array Reliability,* Technical Report, University of California, San Diego, CS92-269, December 1992

[6] Walter A. Burkhard and Jai Menon: *Disk Array Storage System Reliability,* Digest of Papers, The Twenty-Third International Symposium on Fault-Tolerant Computing, p. 432-441, June 1993, Toulouse

[7] Walter A. Burkhard, Kimberly Claffy and Thomas E. Schwarz: *The Balanced Information Dispersal Algorithm,* Proceedings of the Eleventh IEEE Symposium on Mass Storage Systems, October 1991, p. 45-50,Monterey, CA

[8] Walter A. Burkhard, Bruce E. Martin and Jehan-François Pâris: *The Gemini Replicated-File System Testbed*, Information Sciences, volume 48, 1989 , p.119-134

[9] William Feller: **An Introduction to Probability Theory and Its Applications** , volume II, John Wiley Pub., 1968

[10] W. K. Fuchs, K. Wu and J. Abraham: *Low-Cost Comparison and Diagnosis of Large Remotely Located Data Files,* Fifth Symposium on Reliability in Distributed Software and Database Systems, January 1986, 76-73

[11] Garth Alan Gibson:*Redundant Disk Arrays: Reliable, Parallel Secondary Storage,* Ph.D.Thesis, University of California, Berkeley December 1990 (Report UCB/CSD 91/613)

[12] Garth A. Gibson, Lisa Hellerstein, Richard M. Karp, Randy H. Katz and David A. Patterson: *Failure Correction Techniques for Large Disk Arrays,* Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS III), Boston, April 1989, p. 123-132

[13] John L. Hennessy and David A. Patterson: **Computer Architecture A Quantitative Approach,** Morgan Kaufmann Publishers, 1990

[14] Mark Holland: On-Line Data Reconstruction in Redundant Disk Arrays, Ph.D. Thesis, Carnegie Mellon University 1994

[15] Ehud D. Karnin, Jonathan W. Greene and Martin E. Hellman: *On Secret Sharing Systems,* IEEE Transactions on Information Theory, 1983, vol. 29, p. 35-41

[16] Randy H. Katz, Garth A. Gibson and David A. Patterson: *Disk System Architectures for High Performance Computing,*Proceedings of the IEEE, volume 77, 1842-1858, December 1989

[17] Donald E. Knuth: **The Art of Computer Programming, Volume 1 / Fundamental Algortihms.** Second Edition, Addison-Wesley Publishing Company, Reading Massachusetts, 1973

[18] Donald E. Knuth: **The Art of Computer Programming, Volume 2 / Seminumerical Algorithms.** Second Edition, Addison-Wesley Publishing Company, Reading Massachusetts, 1981

[19] Donald E. Knuth: **The Art of Computer Programming, Volume 3 / Sorting and Searching.** Addison-Wesley Publishing Company, Reading Massachusetts, 1973

[20] Edward D. Lazowska, Zahorjan, G. Scott Graham and Kenneth C. Sevcik: **Quantitative System Performance: Computer System Analysis Using Queueing Network Models ,** Prentice-Hall Publishers, Englewood Cliffs, 1984

[21] Edward K. Lee, Peter M. Chen, John H. Hartman, Ann L. Chervenak Drappeau, Ethan L. Miller, Randy H. Katz Garth A. Gibson and David A. Patterson: *RAID-II: A Scalable Storage Architecture for High-Bandwidth Network File Service,* University of California, Berkeley,1992, UCB/CSD 92/672

[22] Jai Menon and Richard L. Mattson: *Distributed Sparing in Disk Arrays,* Proceedings of the COMPCOM Conference, San Francisco, February 1992, p. 410-416

[23] Jai Menon and Dick Mattson: *Performance of Disk Arrays in Transaction Processing Environments,* Proceedings of the 12th International Conference on Distributed Computing Systems, Yokohama, p.302-309, June 1992

[24] J. Metzner: *A Parity Structure for Large Remotely Located Data Files,* IEEE Transactions on Computers Vol C -32, No. 8, 1983

[25] R. Muntz and John C.S. Lui: *Performance Analysis of Disk Arrays Under Failure,* Proceedings of the 16th VLDB Conference, Brisbane, June 1990, p. 162-173

[26] Florence Jesse MacWilliams and Neil James Alexander Sloane: **The Theory of Error-Correcting Codes,** North Holland, 1978

[27] Jehan-François Pâris: *Voting with Witnesses: A Consistency Scheme for Replicated Data,* Proceedings of the 6th International Conference on Distributed Computing Systems, Cambridge, May 1986, p. 606-612

[28] David A. Patterson, Garth A. Gibson and Randy H. Katz: *A Case for Redundant Arrays of Inexpensive Disks (RAID),* Proceedings SIGMOD International Conference on Data Management, Chicago, 1988, p. 109-116

[29] Franco P. Preparata: *Holographic Dispersal and Recovery of Information,* IEEE Transactions on Information Theory, 1989, vol. 35, p.1123-1124

[30] Michael O. Rabin: *Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance,* Journal of the Association for Computing Machinery, 1989, p. 335-348, vol. 36

[31] Michael O. Rabin: In "Sequences, Combinatorics, Compression, Security and Transmission" Springer-Verlag, 1990 , p. 406-419

[32] Martin Schulze, Garth A. Gibson, Randy H. Katz and David A. Patterson: *How Reliable is a RAID?,* Proceedings of the COMPCON Conference, San Francisco, 1989, p. 118-123

[33] Thomas J.E. Schwarz and Walter A. Burkhard: *RAID Organization and Performance,* Proceedings of the 12th International Conference on Distributed Computing Systems, Yokohama, June 1992, p. 318-325