Reliability of a Clustered File System

Nate Dire
<ndire@isilon.com>

IND 526
August 2006

Abstract

This document evaluates the reliability of a particular clustered file system model. A basic knowledge of probability and reliability is assumed; advanced techniques are introduced before application. The model considered here is that of a homogeneous group of servers, each containing the same number of drives, with node-wise redundancy, effectively a repairable k-out-of-n:G system. Two methods of evaluation are compared for accuracy.

Contents

1 Introduction

The explosive growth of information in the Internet Age and advances in clustered computing in the 1990's combined to create a new storage system concept: clustered storage. This trend is characterized by systems comprised of multiple independent, replaceable units which are combined with redundancy to make a robust and scalable system.

The clustered storage industry advertises many benefits to clustered storage, including availability, scalability, and cost. The benefit being considered in this paper is reliability. Clustered storage is effectively an extension of Redundant Arrays of Inexpensive Disks (RAID), a concept introduced in the late 1980s and now quite common in server storage. The basic design principle is that relatively cheap components can be combined with redundancy to create a cost-effective system with greater reliability than a more complex and typically much more expensive monolithic solution.

Evaluating the reliability of clustered storage is not simple. Repair and redundancy add significant complexity to system reliability analysis. When there is no redundancy, failure is defined by a single event. With redundancy, failure is defined by some finite series of events. When repair is included, failure is defined as the proximity of failure events.

The intent of this document is to provide a self-contained introductory survey of techniques for the practicing engineer. There is no original analysis being presented, but rather the application of multiple existing techniques to a specific set of conditions. Only a basic understanding of probability and reliability is assumed; the more advanced techniques are introduced before application.

2 Overview of Clustered Storage

To motivate this discussion of clustered storage, we first consider the traditional architectures which preceded it. The simplest example is large servers with RAID arrays. In each server is an array of disks with some level of redundancy. When more storage is needed, a new server is purchased, and the administrator creates a new file system which is independent of the existing ones. Although the RAID system in each server is quite reliable, the server itself is a single point of failure. Eventually, the administrator has several independent "islands" of storage.

Clustered storage is the logical response to the difficulties of managing traditional storage architectures and several trends:

> **Cheap PC components**. Every year the cost per MB for drives and the cost per MIPS falls. This works against large monolithic systems with custom hardware which must race to keep up with the advances in PC components. It favors putting more complexity in software that can take advantage of better off-the-shelf components as they become available.
>
> **Maturing of RAID technology**. In the 1980's, RAID concepts were relatively new, so designs were relatively limited. As the technology has matured, engineers have found new applications for the ideas and new ways to extend the design. For example, at one point redundancy was either mirroring or "+1" to denote parity

protection. Now many systems support an arbitrary extension of parity. This is further discussed in section 2.2.

The need for scalable storage. Although hard drive capacities increase every year, they don't match the increase in storage needs. Enterprises have ever growing storage needs and want the storage on demand; if they need 100TB now and 200TB next year, they want to add storage in the increments they need, not in the large blocks typical of traditional storage systems.

There are many examples of large, well-known Internet services which employ clustered storage technology. Google's file system is a perfect example of using software to combine thousands of cheap servers into a single, reliable storage system, see [3]. Google's system has a much different architecture from the one under consideration here, however, and there are a number of different storage system architectures which can be considered clustered storage. Some are *heterogeneous*, with components that serve particular purposes, while *homogeneous* architectures have functionally identical components. The system can include a file system software and present files and directories to clients, or provide a small number of volumes or partitions on top of which clients use a distinct file system. There are also variations in how the cluster is organized. Nodes may be individual blades within a larger enclosure or independent, rackable units.

The major design consideration in this context is how the redundancy is structured. A redundancy set is a grouping of components which share redundant information. In the case of mirroring, each component which contains a mirror of a given piece of data is in the same redundancy set. This is more complex for parity protection, but it's important to note that redundancy sets may be *partitioned*. Without partitioning, the redundancy set for a given piece of data may be any grouping of components in the system. With partitioning, the redundancy sets are limited to certain combinations. For example, a partitioning of 10 nodes might create two redundancy sets of 5 nodes. Without partitioning, there are $\binom{10}{5}$ redundancy sets.

2.1 Abstract Model

For this work, we consider a particular model of clustered storage system. This model includes homogeneous nodes which contain multiple drives. Each node is an independent unit capable of providing a full file system interface to clients. The nodes attached to a network where file sharing protocols such as NFS, CIFS, and FTP are used to access the data. This is referred to as network-attached storage (NAS), and a system block diagram is shown in figure 1. The following sections further refine this model.

2.2 Redundancy

Redundancy forms the basis for claims of clustered storage system reliability. The basic idea is identical to that used for RAID. A k-out-of-n system can be built from n components by storing redundancy information.

The simplest approach to redundancy is to use mirroring. Multiple components store an identical copy of a piece of data. If any given copy becomes lost or unavailable, the

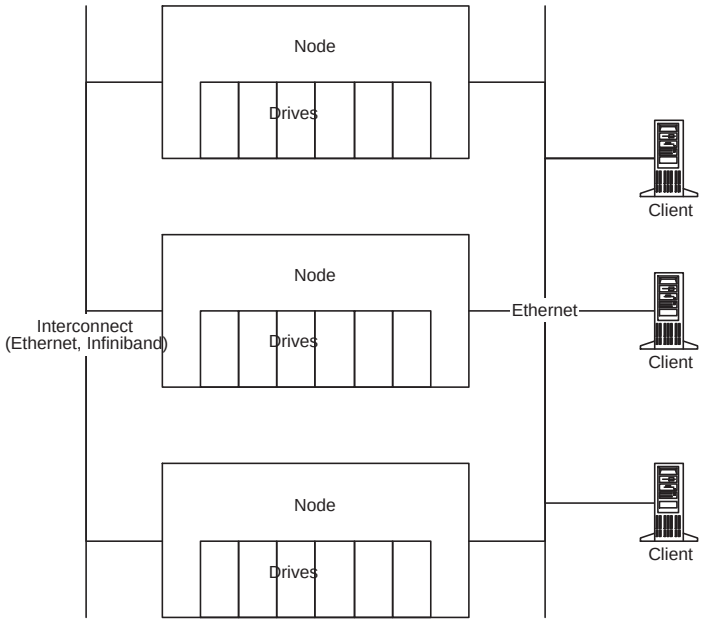

Figure 1: NAS Clustered Storage System

remaining copies may still be accessed. If the mirroring level is n, then mirroring provides a 1-out-of-n system.

Although mirroring is simple, it creates significant storage overhead. In the common case, redundant information is not accessed. There are much more efficient schemes which require extra work to rebuild and calculate the redundancy information. Such schemes employ *error-correcting codes* to create redundancy. With error-correcting codes, the overhead of redundancy can be reduced. For n data units, these schemes can create m additional units of data, such when units are lost, any n of the $n+m$ units can be used to reconstruct the data. This yields a n-out-of-$n+m$ system, where m can be increased to improve reliability, at the cost of calculation overhead and additional space. A good reference for the engineer can be found in [6].

In this system model, redundancy information is striped across nodes. Data is broken up into *stripes* which contain some range of the data and the redundancy information for that data. The redundancy level is characterized by m, which is the number of failures which can be tolerated without data loss. The stripes are of size $n+m$, and require $n+m$ distinct nodes. For example, to protect data at $6+3$, nine nodes are required. This results in the redundancy information adding 50% overhead.

These redundancy sets are not partitioned. If data is protected at $8+2$ on a system with 20 nodes and two nodes fail, there may exist a redundancy set which includes both those nodes. The only limit to the redundancy sets is that each unit must be stored on a

separate node. Thus, drives within the same node never share a redundancy set unless they store information for the same unit.

2.3 Failure as Data Loss

There are a number of ways to interpret the reliability of a storage system since "failure" can be defined quite broadly. A storage system can fail for all the same reasons as any complex computer system. For this work, we want to capture the critical failures while keeping the problem tractable. Since the main purpose of a storage system is to store data, we will define failure as file system *data loss* due to failed storage components.

Data loss occurs when enough components stop functioning that data stored in the file system becomes unrecoverable, that is, when more than m components in a $n + m$ redundancy set fail. When any component fails, a repair process must be started to restore the redundancy. In the following sections, component failures and the process of repair are defined.

2.4 Component Failures

A clustered storage system is composed of many independent hardware components, including drives, CPUs, interconnect, and cooling fans. Enumerating all component failure rates is beyond the scope of this paper. Some simplification is required. In this paper, node and drive failures are the focus.

2.4.1 Drives

Drives are the most numerous logically separable component in the system. A single node may have anywhere from 8 to hundreds of drives. Drives are the most important component of the system since they are the stable storage for the data; when they are lost data is lost. Drives are also one of the least reliable components owing to their rotating mechanical assembly and delicate magnetic media.

Drive manufacturers usually quote MTTFs in the range of hundreds of thousands of hours. These numbers assume replacement before the wear-out period. It's important to note that this document assumes replacement at the manufacturer's recommended time period.

Drive reliability is a rich subject which is highly dependent upon the data made available by manufacturers. There are multiple classes of drives with varying levels of cost, performance, and reliability. The reader is encouraged to investigate this subject separately. In the context of this paper drives are assumed to have a failure rate of approximately 1% per year.

2.4.2 Nodes

Node failure captures the failure of all the other components of the system. These components are similar to any personal computer system: CPU, fans, power supply, motherboard, NICs, and more. With the exception of the fans, most of these components are solid-state and therefore usually more reliable than drives.

Also, these components are not responsible for the stable storage of data, and therefore can typically be replaced without losing data. In the context of this paper, the node failure rate is intended to capture all the components which require replacement of the node. Field-replaceable node sub-components typically impact availability of the data on the enclosed drives, but not the reliability.

In practice, it may be possible to preserve drive data, even when a node must be replaced. This reduces the effective node failure rate to be the failure rate of components which impact drive integrity. This effect is not addressed in this paper.

2.5 Repair

When a component fails, file system data can be rebuilt using the remaining space in the cluster. The process of repair is generally scalable; more nodes in the cluster provides more system throughput for rebuilding data. This repair process typically completes in a matter of hours. This is faster than traditional storage architectures since all the hardware in the cluster is employed to effect the repair.

The process of repair restores the specified level of redundancy. At some later point, the failed component may be replaced with a new unit. For drives, this has little effect on the system. For nodes, the effect can be more dramatic since so much space must be used to store information from the failed node. In practice, the process of repair would need to wait for node replacement before rebuilding the redundancy, but this effect is ignored here.

3 Methodology

Evaluating the reliability of a repairable system is a complex task. It depends on a number of assumptions which may significantly effect the outcome. This section introduces the concepts, notations, and assumptions which characterize the analysis.

3.1 Reliability Concepts

The reliability function for a system, denoted by $R(t)$, indicates the probability of the system functioning at time t. The failure distribution function is denoted by $f(t)$, and represents the p.d.f. for t. Evaluating the reliability of a system typically means characterizing $R(t)$ (or $f(t)$ since there is a known relationship).

In most contexts, it is convenient to characterize the reliability of a system with a single number, rather than a distribution function. The typical number presented is the expected value of $f(t)$, which is the average, or mean, value of t. This is often referred to as the mean time to failure (MTTF). For system with perfect repair, MTTF is the same as the mean time between failures (MTBF).

Although it is convenient for the layman, characterizing the reliability using the MTTF can be misleading. For example, people frequently expect that $R(MTTF) = 0.5$. That is, that the mean is also the median. This is highly dependent on $f(t)$. For example, for the normal distribution, the mean is equal to the median, but for the exponential distribution, the mean represents the 63.2 percentile. Depending on the context it may be more effective

to characterize the reliability in terms of the 10^{th} percentile or reliability at some time t, such as five years.

This paper assumes a constant failure rate. With a constant failure rate, $f(t)$ is an *exponential distribution*, shown by $f(t) = \lambda e^{-\lambda t}$. This allows a single parameter, λ, the failure rate, to characterize the reliability. Since $\lambda = 1/\mathrm{E}[t]$, converting between the failure rate and mean is easy. The rest of the paper will be concerned with characterizing the reliability with the MTTF of the system.

3.2 Notations

The literature on this subject frequently presents a variety of notations, and this system in particular must be characterized by certain parameters. Below are the mathematical notations that will be used throughout the document.

G	RAID group size
m	Redundancy level, or number of failures that can be tolerated
d	The number of drives per node
N	The number of nodes in the cluster
λ_d	The failure rate of drives
λ_N	The failure rate of nodes
μ_d	Drive repair rate
μ_N	Node repair rate

3.3 Failure Model

In order to characterize system reliability, the reliability of the individual components must be characterized. The failure rate of electronic components is typically characterized by a "bathtub curve". In this model, the initial failure rate is high due to manufacturing defects. This is followed by a period of constant failure rate representing the inherent reliability of the product. A time goes by the product suffers wear-out and the failure rate starts to increase.

Organizations try to avoid the higher failure rates at the ends of the bathtub curve, which can prove disastrous for system reliability. The birth end of the curve is typically addressed by a "burn-in" process which attempts to trigger failures due to manufacturing defects. The wear-out period is avoided by limiting the warranty or providing some incentive for replacement.

For this paper, the failure rate is assumed constant. This is reasonable for a clustered storage system since the ends of the bathtub curve are relatively easy to avoid. For burn-in, the system can be given simulated load for 24-48hrs on-site via built-in software. In addition to catching manufacturing defects, this process also finds problems in shipping. Wear-out is less of a concern since computer systems become obsolete quickly. Furthermore, with much of the complexity in software, the hardware is relatively cheap to replace.

The failure model being used here also assumes that failures are detected. This is referred to as *fail-stop* behavior, since the failed device stops performing its function. This is in contrast to "Byzantine" failures, where devices may produce any possible output.

Byzantine fault-tolerance is considered beyond the scope of this work, and is typically not a significant concern in systems such as the one under consideration.

When characterizing the reliability of the clustered storage system, the MTTF of the system will be described as mean time to data loss (MTTDL), to help distinguish it form individual component MTTFs.

3.4 General Assumptions

Any complex analysis relies on a simplifying set of assumptions. This sections lists the assumptions made here, along with some justification.

- *Component failure is detectable.* This is mentioned in the previous section. In this clustered storage model, component failures are not allowed to cause arbitrary behavior (called *Byzantine failure*). In practice, it is rare for devices to exhibit true Byzantine failure, though there is some measurable chance of devices introducing single bit errors. These bit errors can be converted into detectable errors using checksums, or detected and corrected using error-correcting codes.

- *Failures are i.i.d.* This is not the case for several reasons. The failure of one component causes a repair process to start which increases load on the remaining components. Hard drives in particular tend to be vulnerable to environmental factors, which represent a common cause (covered in the next section).

- *Repair time is constant.* Repair time is determined by a number of installation-specific characteristics, including load, cluster size, and space used. Repair time is probably better modeled as a log normal distribution. Instantaneous repair is very unlikely; there is a band of high probability based on the number of nodes, size of drives, and hardware performance; finally, there is a long tail representing all the various unlikely conditions that may delay repair. For this analysis, we assume that in practice, installation-specific parameters can be incorporated by using observed repair times, since any reasonably large installation will have an all-component MTBF on the order of months.

- *Repair is perfect.* In this clustered storage system model, failed components are replaced with new components. These standby components suffer no degradation, and are assumed equivalent to other components in the system. In practice, standby components may not be identical, since computer system components become obsolete so fast.

3.5 Ignored Factors

Some behaviors of this clustered storage system are difficult to quantify or require analysis beyond the scope of this document. In any reliability analysis, it's important to identify which factors are *not* being considered so that estimates can be properly qualified.

The following factors are ignored in this paper, though there may be significant research into quantifying the effects. The reader is encouraged to research these topics separately.

- *Disk sector failures.* Healthy disk drives sometimes report individual sector read failures. This can cause data loss of just the sector, which is just 512B, or could make the file system unusable if the sector holds crucial metadata.
- *Environmental Problems.* Environmental problems are a "common cause" that can cause multiple nodes to fail. This violates the independence of component failures. This is mostly outside the system designer's control, but is an important consideration for any organization.
- *User error.* Issues such as misconfiguration, improper maintenance and vandalism are legitimate threats to reliability of any system. These threats are highly dependent on the implementation and generally difficult to quantify.
- *Software defects.* Software problems can cause a clustered storage system to fail. The file system may become corrupt, or some critical functionality may stop working. The effects of software defects are highly dependent upon the implementation. There is significant research on this subject.
- *Partial Repair.* With double redundancy, it's possible for a component to fail during the repair of another component.
- *Physical replacement delay.* In this clustered storage model, all the space in the system can be used to repair the loss of a component, so no physical replacement or standby component is required to effect repair. However, any reasonably utilized system will quickly run out of space without replacement. It's assumed here that the replacement interval is much less than the overall failure rate of components.

4 Approaches to Evaluating Repairable k-out-of-n Systems

Evaluating the reliability of a non-repairable k-out-of-n:G system with identical components can be straightforward. When replacement is not considered at all, the reliability can be easily expressed as the probability of more than k components functioning:

$$r(k,n) = \sum_{i=k}^{n} \binom{n}{i} R^i (1-R)^{n-i}$$

When the system contains multiple component types, and components may be repaired, the system reliability is not as easy to evaluate. This section presents three basic approaches to evaluating such a system.

4.1 Direct Analysis

Most approaches to analyzing system reliability involve enumerating the various combinations of failures and evaluating the individual probabilities. This quickly becomes intractable for a system with several components, and especially so in the context of clustered storage where literally thousands of hard drives may be involved. Since many of the components are identical, there can be some simplification, but direct analysis remains difficult.

In this paper, direct analysis of a simple RAID system is presented in section 5, which is the only analysis of this form presented.

4.2 Markov Analysis

Complex systems may be difficult to evaluate by enumerating the various combinations of failures. Sometimes it is easier to look at particular states and the transitions between them. This approach is referred to as Markov analysis.

A *Markov process* is random process wherein the transition between states only depends on the current state. If I is the set of states, then a given state i has some probability p_{ij} of transitioning to each state j in I. A Markov state diagram shows the possible states and their transitions.

For example, consider a discrete system with two states, 1 and 2. The system transitions between states in a series of discrete steps. In state, 1, there is a probability 1/2 of transitioning to state 2, and similarly and probability 1/4 of transitioning to state 1 from state 2. Then the stochastic probability matrix for this system is:

$$\mathbf{P} = \begin{pmatrix} 1/2 & 1/2 \\ 1/4 & 3/4 \end{pmatrix}$$

This matrix form allows for use of linear algebra methods in evaluating statistical properties. This example was a case of a discrete Markov chain, where transitions occur in a series of steps. For repairable system reliability we are concerned with time as a continuous value, but the same principles will apply. For further general discussion of Markov chains methods see [4].

4.3 Monte Carlo Simulation

Probalistic analysis of a repairable k-out-of-n grows ever more complex as new components and behaviors are included. State transitions may no longer have the Markov property. For some combinations of failure distributions and components, a closed-form solution for the MTTDL may not even be possible.

Monte Carlo simulation is well suited for more complex systems. The reliability engineer is free to incorporate any number of states and any interactions between components. Furthermore, any properties of the random variable can be recorded without having to perform new analysis.

4.3.1 Theoretical Basis

Let $\mathbf{X} = (X_1, \ldots, X_n)$ denote a random vector having a given density function $f(x_1, \ldots, x_n)$ and suppose we are interested in determining the expected value of a function $g(\mathbf{X})$. Suppose it is not analytically possible to compute or even approximate the expectation.

One way of approximating $E[g(\mathbf{X})]$ is by simulation. Generate a $\mathbf{X}_i$ and compute $Y_i = g(\mathbf{X}_i)$. Do this r times, and by the strong law of large numbers,

$$\lim_{r \to \infty} \frac{\sum Y_i}{r} = E[Y_i] = E[g(\mathbf{X})]$$

Central to Monte Carlo simulation is the ability to simulate a random variable with a specific distribution. In this case, the lifetime of drives is often assumed to be an exponential distribution, yet we generally think of producing uniformly distributed random variables. To produce a random variable with a given continuous distribution from a uniform random variable, we use the following proposition as defined in [8]:

> Let U be a uniform $(0, 1)$ random variable. For any continuous distribution function F, if we define the random variable X by
>
> $$X = F^{-1}(U)$$
>
> then the random variable X has distribution function F.

We can employ Monte Carlo simulation by letting $g(x)$ be the failure p.d.f. for the time to data loss. Then,

$$\text{MTTDL} = E[f(\mathbf{T}_{DL})]$$

4.3.2 Implementation

A Monte Carlo simulator for this clustered storage system is similar to many event-based simulations. Future events are generated randomly. The simulation moves forward to the next event in time, effects that event in the simulated objects, and generates more events. The simulation may end after a period of time, or for this application, at system failure.

```
for each component:
    queue.insert(random_failure(component))
while true:
    event = queue.pop()
    if is_failure(event):
        if data_loss:
            break
        else:
            queue.insert(repair(event.component))
    if is_repair(event):
        queue.insert(random_failure(event.component)
```

Figure 2: Psuedo-code for Monte Carlo simulation

Figure 2 shows psuedo-code for a Monte Carlo simulation of the clustered storage system under consideration. This algorithm produces one sample for t_{DL}. This algorithm can be run multiple times and averaged to produce the MTTDL. Additional statistical properties of t_{DL} can also be generated from these samples, such as variance or percentiles.

Choosing an appropriate number of runs can be difficult. One approach is to specify a precision and run simulations until the desired precision is reached. This can be accomplished using the relatively uncertainty in the parameter, which is:

$$\frac{\sigma}{\bar{t}\sqrt{n}}$$

where σ is the standard deviation, $\bar{t}$ is the expected value of t, and n is the number of iterations. This is explained in [1]. This simulator was implemented and will be used to provide independent verification of Markov analysis.

5 RAID Reliability

As RAID systems are the basis for clustered storage, the reliability of RAID systems also forms the basis for clustered storage reliability. Section 6 will then apply these results to evaluate the reliability in limited cases, and thus validate more complete results.

In the context of this section, a RAID system is a group of G disks with some amount of redundancy m, where $m+1$ failed disks within the group indicate data loss. RAID systems are typically implemented with 8-16 disks, where the redundancy stripes range from 7+1 to 14+2.

5.1 Direct Analysis

Using direct analysis we can calculate the MTTDL_{+1}. Note that if we assume data loss is exponentially distributed, then this single parameter is sufficient to characterize the reliability of the system. Since there is no time-dependency in the calculation, data loss is assumed to be exponentially distributed.

The following argument is presented in [5]. It is reproduced here using this paper's notation and extended for additional redundancy levels. Let G be the number of disks in the redundancy set. We first assume independent and exponential failure rates. This model uses a biased coin with the probability of heads being the probability that a second failure will occur with the MTTR of a first failure. Since disk failures are exponential,

$$\begin{aligned}
\text{P(failure during repair)} &= 1 - \text{P(no failures during repair)} && (5.1)\\
&= 1 - R_1(1/\mu_d)R_2(1/\mu_d)\cdots R_{G-1}(1/\mu_d) && (5.2)\\
&= 1 - R_d(1/\mu_d)^{G-1} && (5.3)\\
&= 1 - (e^{-\lambda_d/\mu_d})^{G-1} && (5.4)
\end{aligned}$$

In any practical scenario,

$$\mu_d \gg \lambda_d \cdot G$$

and since $(1 - e^{-x}) \approx x$ for $0 < x \ll 1$

$$\text{P(failure during repair)} = \frac{\lambda_d \cdot (G-1)}{\mu_d}$$

Then that on a disk failure we flip this coin

heads $\Rightarrow$ a system crash, because a second failure occurs before the first was repaired.

tails $\Rightarrow$ recover from error and continue.

Then,

$$\begin{aligned}
\text{MTTDL}_{+1} &= \text{E[Time between failures]} \cdot \text{E[flips until heads]} && (5.5)\\
&= \frac{\text{E[Time between failures]}}{\text{P(heads)}} && (5.6)\\
&= \frac{1}{\lambda_d \cdot G} \cdot \frac{\mu_d}{\lambda_d(G-1)} && (5.7)\\
&= \frac{\mu_d}{G \cdot (G-1) \cdot \lambda_d^2} && (5.8)
\end{aligned}$$

This addresses the case of +1 redundancy, that is tolerating one failure. What happens if we add arbitrary redundancy? Fortunately, there is a straightforward way to extend this same argument.

For our +1 analysis, we started with the MTBF of drive failures, and then considered the number of failures which occur before a second failure occurs during the repair. For the general case, we start with the frequency of m simultaneous failures, and consider the number of such cases which occur until there is data loss.

$$\begin{aligned}
\text{MTTDL}_{\text{m+1}} &= \text{E[Time between } m \text{ failures]} \cdot \text{E[flips until heads]} && (5.9)\\
&= \frac{\text{E[Time between } m \text{ failures]}}{\text{P(heads)}} && (5.10)\\
&= \text{MTTDL}_{\text{m}} \cdot \frac{\mu_d}{\lambda_d(G-(m+1))} && (5.11)
\end{aligned}$$

Thus, we can construct the MTTDL for m redundancy:

$$\begin{aligned}
\text{MTTDL}_{\text{m}} &= \frac{1}{\lambda_d \cdot G} \cdot \frac{\mu_d}{\lambda_d(G-1)} \cdots \frac{\mu_d}{\lambda_d(G-m)} && (5.12)\\
&= \frac{\mu_d^m}{\prod_{i=0}^{m}(G-i)\lambda_d^{m+1}}. && (5.13)
\end{aligned}$$

5.2 Markov Analysis

We now apply Markov analysis to RAID storage systems. For $m = 1$, there are three basic states: no failures, one failure, and data loss. Figure 3 shows the state diagram, with these three states numbered 1, 2, and 3, respectively. State 3, data loss, is an *absorbing state* since the system has failed and cannot be repaired.

The stochastic transitional probability matrix (with Δt omitted) is:

$$\mathbf{P} = \begin{pmatrix} 1-G\lambda_d & G\lambda_d & 0 \\ \mu_d & 1-(G-1)\lambda_d-\mu_d & (G-1)\lambda_d \\ 0 & 0 & 1 \end{pmatrix}$$

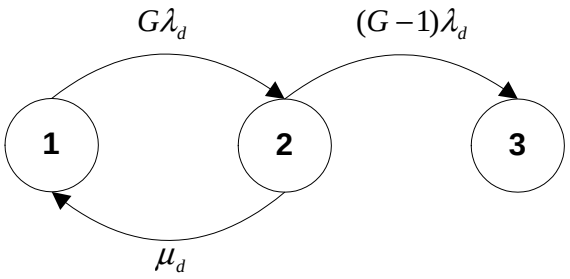

Figure 3: Markov Model for RAID with +1 redundancy

We are primarily concerned with the MTTDL, so we use the method outlined in [1] to calculate it from the probability matrix. First, we create the truncated matrix $\mathbf{Q}$ to account for the absorbing data loss state.

$$\mathbf{Q} = \begin{pmatrix} 1 - G\lambda_d & G\lambda_d \\ \mu_d & 1 - (G-1)\lambda_d - \mu_d \end{pmatrix}$$

Next, we calculate the matrix $\mathbf{M}$, in which each element m_{ij} represents the time spent in state j given the process starts in state i.

$$\begin{aligned} \mathbf{M} &= (\mathbf{I} - \mathbf{Q})^{-1} && (5.14) \\ &= \left(\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - \begin{pmatrix} 1 - G\lambda_d & G\lambda_d \\ \mu_d & 1 - (G-1)\lambda_d - \mu_d \end{pmatrix} \right)^{-1} && (5.15) \\ &= \begin{pmatrix} G\lambda_d & -G\lambda_d \\ -\mu_d & (G-1)\lambda_d + \mu_d \end{pmatrix}^{-1} && (5.16) \\ &= \frac{1}{G(G-1)\lambda_d^2} \begin{pmatrix} (G-1)\lambda_d + \mu_d & G\lambda_d \\ \mu_d & G\lambda_d \end{pmatrix} && (5.17) \end{aligned}$$

Finally, since we only care about cases where the system starts in state 1,

$$\begin{aligned} \text{MTTDL} &= \frac{(G-1)\lambda_d + \mu_d + G\lambda_d}{G(G-1)\lambda_d^2} && (5.18) \\ &= \frac{\mu_d}{G(G-1)\lambda_d^2} + \frac{(G-1)\lambda_d}{G(G-1)\lambda_d^2} + \frac{G\lambda_d}{G(G-1)\lambda_d^2} && (5.19) \\ &= \frac{\mu_d}{G(G-1)\lambda_d^2} + \frac{1}{G\lambda_d} + \frac{1}{(G-1)\lambda_d} && (5.20) \end{aligned}$$

The similarity to equation (5.13) is clear, though additional terms result from the simplifying assumptions made in section 5.

6 Clustered Storage Reliability

The previous section explained how the reliability of a RAID system may be evaluated. Now we consider the reliability of our clustered storage system. First, the RAID equation for MTTDL is applied to nodes and drives separately. Then the combined result is presented based on Markov analysis. These results can be compared to the combined result when $\lambda_d = 0$ and $\lambda_N = 0$, respectively.

6.1 Applying RAID Reliability

First we consider only node failures. In this case, each nodes behaves exactly like a drive in a RAID system. Since the redundancy information is not partitioned, the failure of any $m + 1$ nodes can cause data loss. Thus, the raid group size G is the number of nodes N, and we replace the drive failure and repair rates in equation 5.13 with those of nodes:

$$\text{MTTDL} = \frac{\mu_N^m}{\prod_{i=0}^{m}(N-i)\lambda_N^{m+1}} \tag{6.1}$$

Since there are typically several drives per node, and drives tend to be the more unreliable components by design, a node-only MTTDL is not very useful. Fortunately, the RAID MTTDL equation can be modified to only account for drives in the cluster storage system.

Most drives in the clustered storage system are potentially part of the same redundancy set. Since data is striped across nodes, drives in the same node are *not* in the same redundancy set. That is, multiple drive failures within a node still only count as one failure against the redundancy level of the data (m).

Thus, during repair, only a failure in the nodes outside of those already having a failure can cause data loss. At each failure, the number of nodes which may cause a critical failure are reduced.

$$\text{MTTDL} = \frac{1}{\lambda_d \cdot N \cdot d} \cdot \frac{\mu_d}{\lambda_d(N-1)d} \cdots \frac{\mu_d}{\lambda_d(N-m)d} \tag{6.2}$$

$$= \frac{\mu_d^m}{\prod_{i=0}^{m}(N-i)d^{m+1}\lambda_d^{m+1}} \tag{6.3}$$

6.2 Combined Reliability Model

A Markov analysis of similar storage systems is presented in [7]. A simplified version is shown in figure 5. The diagram is constructed by increasing the number of drive failures along the x-axis of the diagram and increasing the number of node failures along the y-axis of the diagram. The absorbing state, data loss, is indicated by state F and occurs when $m + 1$ nodes are either failed or have a drive failed.

Solving for the MTTDL using matrix methods is beyond the scope of this document. As shown in [7], this matrix yields the following result:

$$\text{MTTDL} = \frac{(\mu_d \mu_N)^m}{\prod_{i=0}^{m} (N-i)(\lambda_N + d\lambda_d)(\mu_d \lambda_N + d\mu_N \lambda_d)^m} \tag{6.4}$$

If this is correct, then it should match equations (6.1) and (6.3) when drive and node failures are ignored, respectively.

First, let us ignore drive failures. Then, $\lambda_d = 0$, and equation (6.4) becomes:

$$\begin{aligned}
\text{MTTDL} &= \frac{(\mu_d \mu_N)^m}{\prod_{i=0}^{m} (N-i)(\lambda_N + d \cdot 0)(\mu_d \lambda_N + d\mu_N \cdot 0)^m} && (6.5)\\
&= \frac{\mu_d^m \mu_N^m}{\prod_{i=0}^{m} (N-i)\mu_d^m \lambda_N^{m+1}} && (6.6)\\
&= \frac{\mu_N^m}{\prod_{i=0}^{m} (N-i)\lambda_N^{m+1}} && (6.7)
\end{aligned}$$

This shows that equation (6.4) matches (6.1) when drive failures are ignored. Now let us ignore node failures, such that $\lambda_N = 0$. Then, equation (6.4) becomes:

$$\begin{aligned}
\text{MTTDL} &= \frac{(\mu_d \mu_N)^m}{\prod_{i=0}^{m} (N-i)(0 + d\lambda_d)(\mu_d \cdot 0 + d\mu_N \lambda_d)^m} && (6.8)\\
&= \frac{\mu_d^m \mu_N^m}{\prod_{i=0}^{m} (N-i)d^{m+1}\mu_N^m \lambda_d^{m+1}} && (6.9)\\
&= \frac{\mu_d^m}{\prod_{i=0}^{m} (N-i)d^{m+1}\lambda_d^{m+1}} && (6.10)\\
& && (6.11)
\end{aligned}$$

Now we see that (6.4) is equivalent to (6.3) when node failures are ignored.

Hence, as we would expect, the Markov analysis is consistent with the direct analysis of the MTTDL. While it is reassuring that two distinct methods produced consistent results, these two approaches rely on similar principles. In the next section, we determine whether equation (6.4) is consistent with results from Monte Carlo simulation.

7 Verification

Since this paper evaluates the reliability of an abstract clustered storage system model, there is no field data to validate the MTTDL estimates. The Markov analysis has been validated against the direct analysis of RAID reliability for limited cases. This section compares specific results from simulation to the Markov analysis to provide some confidence in this analysis. All the assumptions and simplifications are shared between the simulation and the analytic approaches. Thus, there is no validation of those assumption and simplifications.

Figure 4 lists the MTTDL of select scenarios as calculated by both the equation (6.4) and Monte Carlo simulation. The numbers are chosen to be reasonable real-world values and provide reasonable coverage of the input space.

		$1/\lambda_N$	$1/\lambda_d$	$1/\mu_N$	$1/\mu_d$		MTTDL (years)	
N	d	(years)	(years)	(hours)	(hours)	m	Simulation	Equation
10	8	150	75	48	8	1	754	731
10	16	150	75	24	6	1	316	307
20	12	50	50	36	10	2	3015	4432
20	24	50	50	30	5	2	1435	2493
100	16	100	100	40	9	3	1836	6744
100	32	100	100	30	7	3	534	1321

Figure 4: MTTDL estimates from simulation and equation (6.4)

The results are mixed. When $m = 1$, the estimates are quite close. The results for $m = 2$ and $m = 3$, are off by a factor of 2 and 3 respectively. Given the scale of the values, that is, the polynomial impact of m, the values are close enough to validate both approaches. There appears to be some relatively subtle difference which grows with m.

One possible explanation is a difference in the way the simulator and Markov model treat repair under multiple failures. With the Markov model, when repair is interrupted, the system "forgets" any progress made on rebuilding data on existing failed components, and considers all failures to have occurred concurrently. In the simulator, further failures don't adjust the repair time of existing failed components, which more closely matches a real system. This effect is negligible when $m = 1$ because additional failures during repair cause data loss. The effect is multiple with larger values of m because repair can be interrupted multiple times by further failures.

Furthermore, neither model accounts well for multiple drive failures on the same node. In the Markov analysis, this effect is completely ignored. In the simulator, drive failures in the same node have the same repair time. But the failure of one drive loads the system such that additional failures slow the repair rate of each failure.

8 Conclusion and Future Work

This document has surveyed methods of evaluating the reliability of a clustered storage system. Repairable k-out-of-n systems are a challenge for system reliability analysis. Two main

approaches, Markov Analysis and Monte Carlo simulation were introduced and compared. These methods were validated with some limited success.

Future work will need to include refining both approaches and using field data of a particular system to validate the models. The repair process appears to be a challenge in all cases, since certain subtleties of repair are difficult to model. The most important goal, however, is to treat repair consistently so that results can be fairly compared.

References

[1] R. Billinton and R.N. Allan. Reliability Evaluation of Engineering Systems. Plenum Press, New York and London, 1992.

[2] N.B. Fuqua. The Applicability of Markov Analysis Methods to Reliability, Maintainability, and Safety. Selected Topics in Assurance Related Technologies, volume 10, number 2.

[3] S. Ghemawat, H. Gobioff, and S. Leung. The Google File System. Symposium on Operating Systems Principles 2003.

[4] J.R. Norris. Markov Chains. Cambridge University Press, 1997.

[5] D.A. Patterson, G. Gibson, and R.H. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). Proc. Int'l Conf. Management of Data, ACM, 1989, pp. 109-116.

[6] J. Plank. A Tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-like Systems. Available through http://cs.utk.edu/ plank/, February 1999.

[7] K.K. Rao, J.L. Hafner, R.A. Golding. Reliability for Networked Storage Nodes. Technical Report RJ 10358, IBM Research, September 2005.

[8] S. Ross. Introduction to Probability Models. Academic Press; 7th edition, February 2000.

[9] Q. Xin, et al. Reliability Mechanisms for Very Large Storage Systems. 20th IEEE / 11th NASA Goddard Conference on Mass Storage Systems and Technologies, San Diego, CA, April 2003.

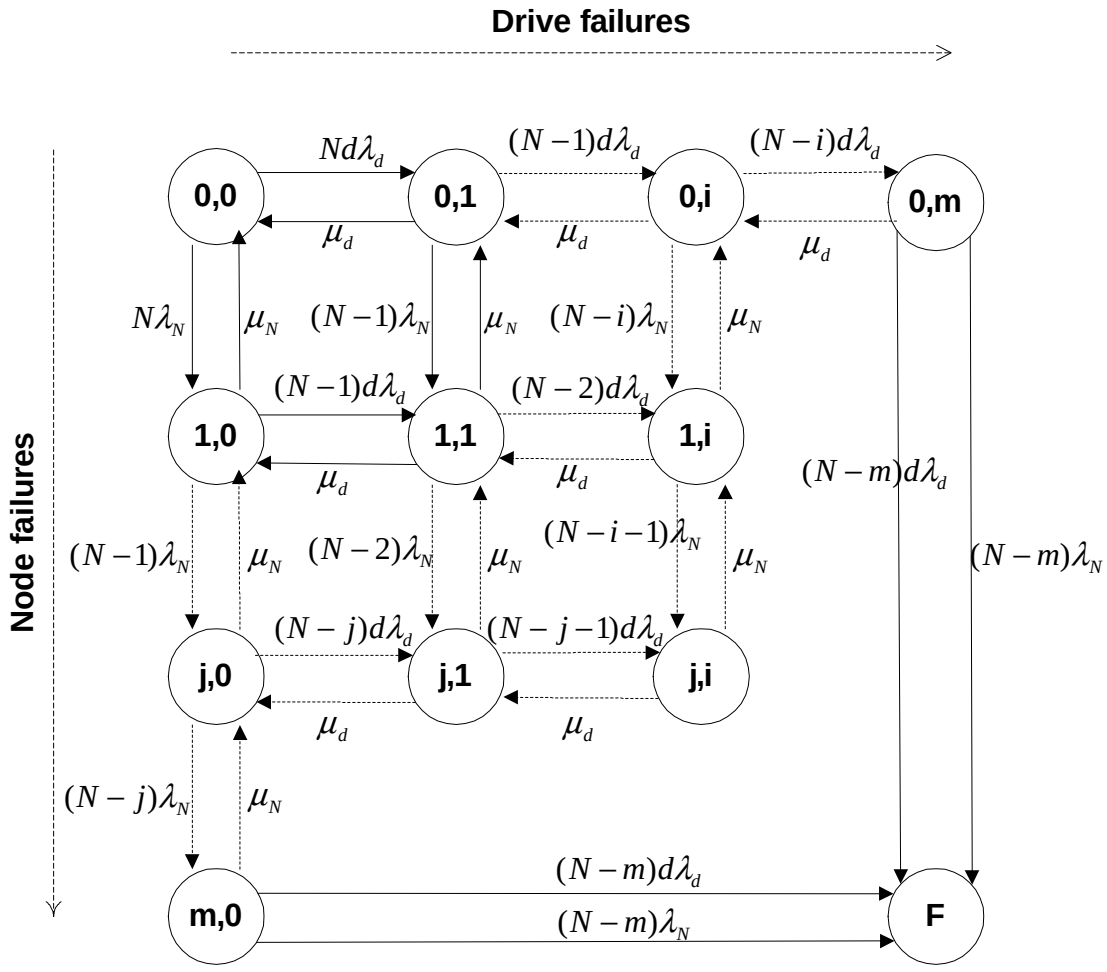

Figure 5: Markov Model for Clustered Storage