Erasure Coded Data Durability

2021-09-24

We calculate the erasure coded data durability in terms of the device failure rates and the recovery speed. We put more emphasis on RAID5&6.

blog: https://tetraquark.vercel.app/posts/raid_durability/?src=pdf

email: quarktetra@gmail.com

Introduction

Whether they are SSDs or HDDs, physical storage devices fail. Once cannot afford to lose data because of such failures, hence the need for redundancy arises. The simplest and the most well-known redundancy is a back up which is a clone of the original data. However, this is highly inefficient since the number of devices is doubled. That is why RAID(Redundant Array of Inexpensive Drives) was invented.

A RAID 5 system creates one parity out of data and stores in on an additional physical drive. The original data with the additional parity bit creates a redundant system which is protected against a single drive failure, i.e., in the case of such a failure, the original data can be reconstructed from the healthy drives. The system typically has an idle drive or user is required to swap the failed drive with a new one so that the data recovery starts immediately. Once the recovery is done, the system is back to its 1-redundant state. As drive capacities reached 20TB, the recovery process takes very long. Consider a failed 20TB drive and a typical recovery rate of 50MB/s. In this case it will take 4.7 days to complete the recovery. What if another drive fails in that time frame? You lose data!

Enter RAID 6, which is a system that creates two pieces of parities to protect against two simultaneous failures. This certainly increases the durability of the data, however certain applications require durability levels that cannot be met by RAID6. In such applications, more redundancy is introduced to create erasure coding.

The math of creating redundancy involves Galois fields and it is a fascinating branch of math! However, I will save that for another post and concentrate on the durability calculations for a given erasure coding. I will want to dive into an aspect which seems to be ignored in most calculators/posts: it is the unrecoverable read error (URE). In this post you will see that UREs effectively kill the last layer of redundancy. In the particular case of RAID6, we will see that its protection is **not** much stronger than RAID5 once the corrupted sectors are included.

I will also show that there exist good models to calculate data durability along with bad and ugly ones. I hope that people will stop using models that are so wrong that they get the answer off by a factor of 10^8!

History of Erasure Coding

Figure 1 shows the evolution of the erasure coding methods from replication to more advanced technologies. The image is taken from a Seagate presentation and has certain Seagate-specific content.

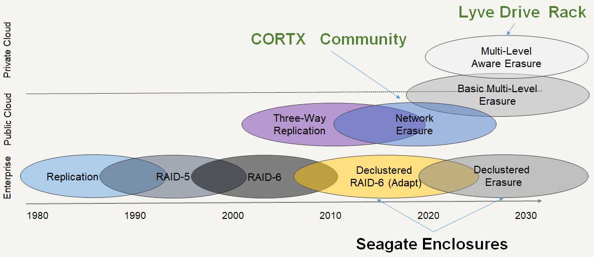


Figure 1: A brief timeline of the erasure coding. Replication has very poor capacity efficieny. RAID5(6) provides protection against one(two) failure(s). Seagate enclosures can be used to declustered parity as well as erasure coding across network. Durability and availablity can be further improved by implementing Multi-layer erasure coding.

The simplest way of creating redundancy is replication, but this has a very poor capacity efficiency.

RAID 5 and Raid 6 introduce parities to protect data against device failures.

More advanced erasure coding schemes include declustered RAID6, and multi-layers.

Creating Redundancy

Simplest example(RAID5): compute and store the parity data.

Given two bytes of data: $B_1 = 01001001$ and $B_2 = 11011010$

The parity data: $P = B_1 \oplus B_2 = 10010011$

Assume the drive that stored B_1 fails:

Compute $P \oplus B_2$, which is equal to B_1.

The data on the failed drive can be reconstructed.

Fault tolerant to one drive failure.

c pieces of redundancy data:

Parities computed using *Reed-Solomon algorithm*.

Fault tolerant to c simultaneous drive failures.

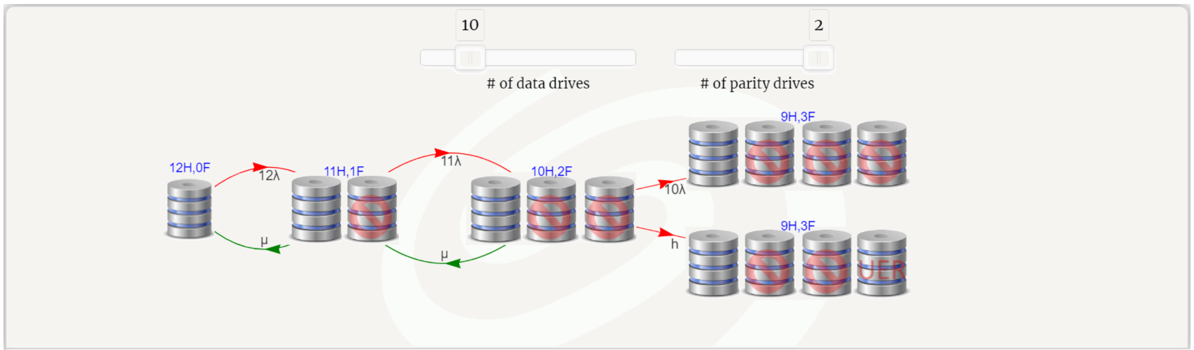
Main Question: What is the probability of having $c + 1$ simultaneous failures?

RAID with URE

?@fig-raiduer shows a symbolic representation of RAID5/6 states and transitions. You can change the sliders to adjust the numbers.

# of data drives

# of parity drives



10

# of data drives

2

# of parity drives

Figure 2: A Symbolic representation of the RAID including URE failures. Use the input sliders to change the number of drives or the redundancy. We will later derive the formulas for data durability for RAID5, and RAID6.

The red arrows represent drive failures. Rate is scaled with the total number of drives:

λ is the failure rate per drive, and $n\lambda$ is the total failure rate for n drives.

The arrow denoted with h represents the data loss due to UREs- to be discussed in more detail later.

The green arrows represent repairs. Failed drives are replaced, and data is rebuilt.

The repair time, $1/\mu$, depends on capacity and the repair rate. It may be as long as several days.

Data is lost when the system moves to the right-most states.

Key metric: Mean Time to Data Loss (MTTDL): it is a function of n, c, λ, μ, drive capacity and UER.

The durability is usually a number which is very close to one. It is typically reported as the **number of nines** which is defined as the instances of leading 9's in reliability. For example 0.998 has 2 nines, 0.9991 has 3 nines. Mathematically it is defined as follows:

$$\text{number of nines} = \text{Floor}\left(-log10(1 - \text{Durability})\right). \tag{1}$$

Let's take a look at the internet and see how people compute the number of nines.

A survey of models

Erasure coding was invented decades ago, and the associated data durability calculations have been well established in the literature. However, this doesn't stop people from inventing their own methods to compute data durability, not that anything is wrong with that! The problem is that, as I will show below, some of these methods are so wrong that they end up being off by several orders of magnitude. Let's get started.

The Wasabi Model

This model is shows up in a white paper from cloud storage provider Wasabi[1]:

When an object is written to Wasabi, the Wasabi erasure coding algorithm converts the object into a series of data and parity fragments, and stores each fragment on 20 different drives. When you read an object from storage, Wasabi reassembles the object using the fragments. In the event one or more drives fail (up to 4 drives), an object can be fully reconstructed using any 16 of the 20 fragments. This means that Wasabi can withstand the failure of up to any 4 drives within a storage slice without losing data. For AFR, Wasabi uses industry-accepted guidelines and assumes a conservative drive AFR of 5%. In practice, our observed AFRs are much lower. For MTTR, Wasabi estimates 3.4 days per drive, based on the calculation below: MTTR = 14 TB drive capacity * 50 MB/s drive write speed = 3.4 days to fully write a replaced 14 TB drive As stated earlier in the erasure coding discussion, in order for an object to be lost, more than 4 drives in a Wasabi storage slice would have to fail. To understand the probability of this, the first step is to understanding the probability a single drive failing using the calculation below: Probability of a 1 drive failing = AFR (5% year) * MTTR (3.4 days) * (1/365 year/days) = 4.66 x 10-4 The next step would be to understand the probability of four drives failing while another drive in the storage slice was rebuilding. This would be a potential data loss scenario because five drives in a storage slice would not be available (one in rebuild mode, plus four new failures). To calculate this probability, this formula applies: Probability of 4 drives failing = Probability of 1 drive failing (4.66 x 10 -4) to the 4th power = 4.7 x 10 -14 The final step in calculating data durability is to factor in the probability of 4 drives failing using the following formula: Data Durability = 1 - (probability of 4 drives failing) = 1 - 4.7 x 10-14 = .99999999999995 As seen in the above calculations, Wasabi storage architecture provides greater than 11 x 9s of durability. The calculated number is actually 13 x 9s but for the sake of taking a conservative approach to the calculations and to align with how most of the hyperscalers position their data durability, Wasabi uses 11 x 9s as the published data durability metric.

—

This model is very problematic! Consider the following generic set up where you can change the numbers if you would like, and follow the Wasabi model:

(n) drives: capacity(C) in TB: redundancy(c): recovery speed (S) in MB/s: AFR in %:

At this recovery rate, the recovery time from a drive failure is: days.

The probability of losing a single drive in days is:

The system will lose data when there are failures in days, which has the probability: () .

Data durability= 1- Probability of failure(s) in days=1- () = .

Why is it wrong? It is because the metric calculated is not the data durability for a system of drives with -redundant EC! It is for 1+ [original+ mirror(s)]. Note that the total number of drives, , did not even enter the equations! There are binomial(,)= combinations to choose failure(s) out of drives. The reported number is not even the durability for over a year, it is just over days. There are such frames in a year. With the binomial coefficients and the # frames/year included, this model over-reports durability by at least nines. Furthermore, if the UREs were included, that would have reduced the durability by 3 more nines (more on that later). Therefore, this model is off by about 8 orders overall.

The BackBlaze Model

This is a model introduced by BackBlaze[2], [3]. Let's look at their Poisson-based model:

For our Poisson calculation, we use this formula:

$$P(k\,\text{events in interval}) = e^{-\lambda}\frac{\lambda^k}{k!} \tag{2}$$

The values for the variables are:

Annual average failure rate = 0.0041 per drive per year on average

Interval or “period” = 156 hours (6.5 days)

Lambda = ((0.0041 * 20)/((365*24)/156)) =0.00146027397 for every “interval or period”

e = 2.7182818284

k = 4 (we want to know the probability of 4 “events” during this 156 hour interval)

Here’s what it looks like:

$$
\begin{aligned}
P(k\,\text{events in interval}) &= e^{-\lambda}\frac{\lambda^k}{k!} \\
P(k\,\text{events in interval}) &= 2.71828^{-0.00146027397}\frac{0.00146027397^k}{4*3*2*1} \qquad (3)
\end{aligned}
$$

If you’re following along at home, type this into an infinite precision calculator:[3]

(2.7182818284^(-0.00146027397)) * (((0.00146027397)^4)/(4*3*2*1))

= (1 – 1.89187284e-13)^56 = (0.999999999999810812715)^56 = 0.99999999999 (11 “nines”)

The sub result for 4 simultaneous drive failures in 156 hours = 1.89187284e-13. That means the probability of it NOT happening in 156 hours is (1 – 1.89187284e-13) which equals 0.999999999999810812715 (12 nines).

But there’s a “gotcha.” You actually should calculate the probability of it not happening by considering that there are 56 “156 hour intervals” in a given year. That calculation is:

= (1 – 1.89187284e-13)^56 = (0.999999999999810812715)^56 = 0.99999999999 (11 “nines”)

[3] The complexity will break most standard calculators.

—

First of all, note that BackBlaze indeed incorporates the number of drives into the model with the effective λ. That is great. The other good thing is that they know that the first calculated number is the durability for a single time frame, i.e., a single repair time window. They extend it to the full year, as illustrated in Figure 3.

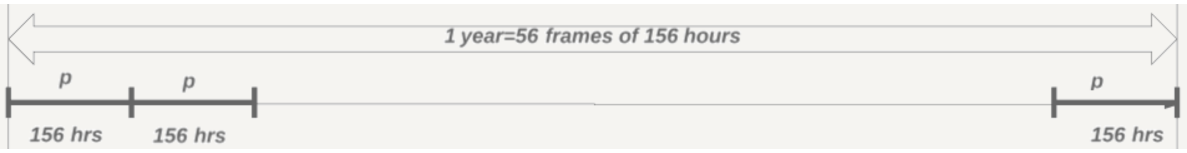


Figure 3: An illustration of the time frames in the BackBlaze model.

The binomial model they construct is similar, and is not fundamentally different than the Poisson model. Since the chosen distribution is not critical for the main point I will argue below, I will re-write their binomial model in a slightly different form which, I believe, will make it easier to follow: Let us first make the input numbers interactive, and repeat the BackBlaze Binomial model:

(n) drives: capacity(C) in TB: redundancy(c): recovery speed $(S\ )$ in MB/s: AFR in %:

The recovery time from a drive failure is: $T \equiv \frac{C}{S}=$ days.

The AFR value can be converted to the failure rate as: $\lambda = -\frac{\ln(1-\text{AFR})}{\text{year}} \simeq \frac{\text{AFR}}{\text{year}}=$.

The probability of a given drive to fail in days is: $p_f \simeq \lambda T=$.

The system will **not** lose data when there are at most failures in days.

The probability of not losing data in days is: $p_{\text{no-loss}} = \sum_{i=0}^{c} \binom{n}{i} (p_f)^i (1-p_f)^{n-i} =$

There are $N_F = \frac{365}{T}=$ such frames in a year.

The probability of not losing data in a year is: $(p_{\text{no-loss}})^{N_F} =$

The BackBlaze model is still simple, but… it has a few issues:

The segmentation of a year into chunks of days implies that data is lost only when failures happen in a given frame. The cases of failures within days but spanning two subsequent frames are missed.

It is implicitly assumed that every frame starts with a -redundant system. In reality, there may be up to ongoing repairs exiting the previous frame, and a single drive failure early in the next frame will cause data loss. The mistake they make is to assume that $p_{\text{no-loss}}$ will be the same at every window, and it clearly is not. In fact, $p_{\text{no-loss}}$ at an arbitrary frame depends on all the previous frames, that is, it is time dependent. We can also say that it will increase as t increases. Therefore one cannot simply raise it to the power of N_F.

Ignoring UREs, BackBlaze model works reasonably well in the low AFR limit (still off by at least factor of ~ 2). If we include UREs, which we should, we can say that BackBlaze model overestimates the durability at the inputs above by nines.

The most intuitive model

Figure 4 shows the time scales in a repairable system.

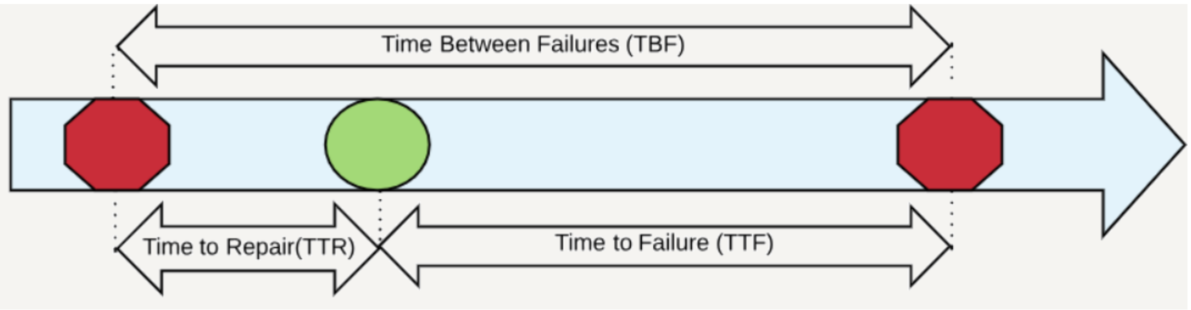


Figure 4: An illustration of the time scales in a repairable system.

A system of n drives has the Mean Time To Failure: $\mathrm{MTTF} = 1/(n\lambda)$.

A failed drive is replaced, and rebuilt. Mean Time To Repair: $\mathrm{MTTR} = 1/\mu$.

Fraction of the time spent for repair: $\frac{\mathrm{MTTR}}{\mathrm{MTTF+\ MTTR}}$

Fraction of the time spent for repair: $\frac{\mathrm{MTTR}}{\mathrm{MTTF+\ MTTR}} \simeq \frac{\mathrm{MTTR}}{\mathrm{MTTF}} = \frac{n\lambda}{\mu}$ (MTTF $\gg$ MTTR).

There are $n-1$ drives left running. The rate of failure: $(n-1)\lambda$

Multiply this rate with the fraction of time in recovery: $\frac{n\lambda}{\mu}(n-1)\lambda = \frac{n(n-1)\lambda^2}{\mu}$

This is the rate of data loss. Inverting the expression: $\mathrm{MTTDL}_1 = \frac{\mu}{n(n-1)\lambda^2}$

When there are two parity drives, we can iterate to get: $\mathrm{MTTDL}_2 = \frac{\mu^2}{n(n-1)(n-2)\lambda^3}$

For c parity drives, we can recursively iterate to get: $\mathrm{MTTDL}_c = \left[\frac{\mu}{\lambda}\right]^c \frac{(n-c-1)!}{\lambda n!}$, which is the standard expression for durability[4].

$c = 1$ is a RAID5, and $c = 2$ is a RAID6 setup. $c \geq 3$ cases are refereed to as Erasure Coding in general.

This model is very intuitive, and works great, but...

It is not clear how to include UREs,

It is not extendable to more complicated problems.

The Markov chain model

This chapter will outline the methodology to calculate the data durability. We will start with a system with no redundancy to introduce the concept of Markov chains, and later will extend the analysis to repairable systems with redundancy. This methodology will enable us to address the UREs.

0 **redundancy**

Just to warm up, let's consider a system with n drives and no redundancy. This means that user data will be lost with any drive failure. What is the reliability of such a system given the reliability characteristics of individual drives? To keep the math simple, let's assume that drives themselves have exponentially distributed failure times, i.e., they fail at a constant rate λ, or equivalently with a certain annual failure rate AFR. Now we are putting n of these drives in a box. How can we describe failures of such systems? The answer is rather obvious: it will still be an exponential distribution with rate $n\lambda$. So, the reliability of such a system is

$$\mathrm{R}(t) = e^{-n\lambda t}. \tag{4}$$

More rigorous math

The probability density and the cumulative probability functions for an exponentially distributed random variable T_i are

$$f(t) = \lambda e^{-\lambda t}, \tag{5}$$

$$F(t) = \int_0^t d\tau f(\tau) = 1 - e^{-\lambda t}, \tag{6}$$

where λ is the failure rate. A system of n storage devices with no redundancy will lose data as soon as a device fails. Formally, we have n independent and identically distributed failure times T_i, $i \in [1, n]$, with density f and the corresponding cumulative distribution F as defined in Eqs. 5 and 6 , respectively. We want to calculate the distribution of earliest failure time, i.e., $T = \min(T_1, T_2, \cdots, T_n)$. We can follow the formal definitions of cumulative probability distribution[5] to construct $F_T(t)$ as follows:

$$\begin{aligned} F_T(t) &= P(T \leq t) = 1 - P(T > t) \\ &= 1 - P\{\min(T_1, T_2, \cdots, T_n) > t\} \\ &= 1 - [1 - F(t)]^n = 1 - e^{-n\lambda t}, \end{aligned} \tag{7}$$

where we make use of the fact that if $\min(T_1, T_2, \cdots, T_n) > t$ then each $T_i > t$. Since there is no redundancy, Eq. 7 is also the cumulative probability density. The reliability can be calculated as $R = 1 - F_T$, which results in Eq. @ref(eq:reli-n-drives).

Just to get a sense of the result, let's plug in some typical numbers: $n = 20$, AFR$= 0.5\%$. Note that for AFR $\ll 1$, $\lambda \sim \frac{AFR}{8766}$ (1/hours), where 8766 is the number of hours in a year. Durability in this case is 0.904, which is unacceptably poor. This is why we need to add redundancy to the system. Before we move onto the systems with redundancy, let's solve this problem one more time in the context of Markov chains. Markov chain is a stochastic model that describes sequence of possible events, such as drive failures and drive restorations where the probability of each event depends on the previous state of the system [6]. Let's draw the Markov chain for the system that we considered earlier, in Figure 5.

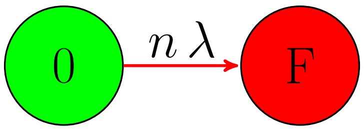


Figure 5: Markov chain for a system with no redundancy.

The red arrow in Figure 5 shows the transition from a healthy $n-$drive state to the failure state. The rate of this event is $n\lambda$. Let's call the probability of the system to be in the $n-$drive state p_n, and the other state p_{n-1}. The differential equations governing the time evolution of the system can be written as

$$\begin{aligned} \dot{p}_n(t) &= -n\lambda p_n(t) \\ \dot{p}_{n-1}(t) &= +n\lambda p_n(t), \end{aligned} \tag{8}$$

which simply show that there is an out-flux from p_n to p_{n-1}. This will be a total overkill, but let's Laplace transform these equations with the initial conditions being $p_n(0) = 1$ and $p_{n-1}(0) = 0$,i.e., we start with $n-$functioning drives. We get:

$$\begin{aligned} sP_n(s) - 1 &= -n\lambda P_n(s) \\ sP_{n-1}(s) &= +n\lambda P_n(t), \end{aligned} \tag{9}$$

where P's show the Laplace transformed functions. Converting this into a matrix equation

$$\begin{bmatrix} s + n\lambda & 0 \\ -n\lambda & s \end{bmatrix} \begin{bmatrix} P_n(s) \\ P_{n-1}(s) \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \tag{10}$$

which is solved by inverting the matrix

$$\begin{bmatrix} P_n(s) \\ P_{n-1}(s) \end{bmatrix} = \begin{bmatrix} s + n\lambda & 0 \\ -n\lambda & s \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{s+n\lambda} \\ \frac{1}{s} - \frac{1}{s+n\lambda} \end{bmatrix}. \tag{11}$$

Finally we inverse Laplace transform to get

$$\begin{bmatrix} p_n(t) \\ P_{n-1}(t) \end{bmatrix} = \begin{bmatrix} e^{-n\lambda t} \\ 1 - e^{-n\lambda t} \end{bmatrix}. \tag{12}$$

So the reliability of the system is

$$\mathrm{R}(t) = 1 - p_{n-1}(t) = e^{-t/\ \mathrm{MTTDL}_0} \text{ with } \mathrm{MTTDL}_0 = \frac{1}{n\lambda}. \tag{13}$$

We went through excessive math, but this is basically the procedure for analyzing a generic Markov chain: Plot the chain with incoming and outgoing arrows, construct the transition differential equation, Laplace transform and convert to a matrix form, invert the matrix and transform back to time domain.

1 redundancy

Let's apply this to an $n-$drive system with one redundancy, which will have the Markov chain in Figure 6.

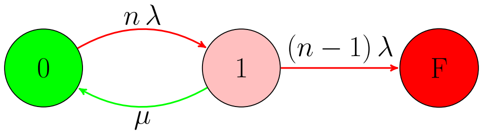


Figure 6: Markov chain for a system with one redundancy.

The red arrows in Figure 6 represent drive failures, the rate of which scales with the number of healthy drives, and the green arrow shows rebuild of drives with a rate that we will call μ. The transition equations:

$$\begin{aligned}
\dot{p}_n + n\lambda p_n - \mu p_{n-1} &= 0 \\
\dot{p}_{n-1} + (n-1)\lambda p_{n-1} + \mu p_{n-1} - n\lambda p_n &= 0 \\
\dot{p}_{n-2} - (n-1)\lambda p_{n-1} &= 0.
\end{aligned} \tag{14}$$

After Laplace transforming and converting to a matrix equation, we get:

$$\begin{bmatrix} P_n(s) \\ P_{n-1}(s) \\ P_{n-2}(s) \end{bmatrix} = \begin{bmatrix} s+n\lambda & -\mu & 0 \\ -n\lambda & s+(n-1)\lambda+\mu & 0 \\ 0 & -(n-1)\lambda & s \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}. \tag{15}$$

We are interested in p_{n-2}, which is the probability that system will have 2 simultaneous failures, hence data will be lost. Inverse Laplace transforming, we can calculate the reliability as

$$\begin{aligned}
\mathrm{R}(t) &= 1 - p_{n-2}(t) \\
&= \frac{\mu + (2n-1)\lambda + \sqrt{(\lambda-\mu)^2 + 4\lambda\mu n}}{2\sqrt{(\lambda-\mu)^2+4\lambda\mu n})} e^{-\frac{1}{2}\left(\mu+\lambda(2n-1)-\sqrt{(\lambda-\mu)^2+4\lambda\mu n}\right)t} \\
&- \frac{\mu + (2n-1)\lambda - \sqrt{(\lambda-\mu)^2 + 4\lambda\mu n}}{2\sqrt{(\lambda-\mu)^2+4\lambda\mu n})} e^{-\frac{1}{2}\left(\mu+\lambda(2n-1)+\sqrt{(\lambda-\mu)^2+4\lambda\mu n}\right)t} \\
&\sim \left[1 + n(n-1)\frac{\lambda^2}{\mu^2}\right] e^{-\frac{t}{\mu/[\lambda^2 n(n-1)]}} - n(n-1)\frac{\lambda^2}{\mu^2} e^{-t[\mu+\lambda(2n-1]},
\end{aligned} \tag{16}$$

where we expanded in powers of λ/μ and kept the leading terms in the coefficients and exponents. We can actually further simplify the equation since $\frac{\lambda^2}{\mu^2} \ll 1$, for a typical storage device, we can use AFR $< 1\%$ and the restore time is at the order of days, which gives $\frac{\lambda}{\mu} \sim 10^{-4}$. Therefore, we can safely drop $\frac{\lambda^2}{\mu^2}$ terms to get

$$\mathrm{R}(t) \sim e^{-t/\ \mathrm{MTTDL}_1} \text{ with } \mathrm{MTTDL}_1 = \frac{1}{n(n-1)\lambda}\frac{\mu}{\lambda}. \tag{17}$$

2 **redundancies**

It will get more complicated but, we will be getting better at extracting the relevant term. Let's apply this to an $n-$drive system with two redundancies, which will have the Markov chain in Figure 7.

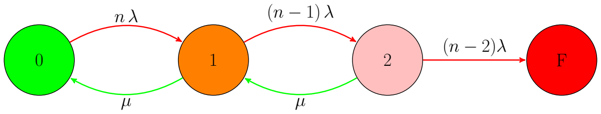


Figure 7: Markov chain for a system with two redundancies.

The state equation in the s-domain is

$$\begin{bmatrix} P_n(s) \\ P_{n-1}(s) \\ P_{n-2}(s) \\ P_{n-3}(s) \end{bmatrix} = \mathcal{S} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \tag{18}$$

where we defined $\mathcal{S}$ as:

$$\begin{bmatrix} s+n\lambda & -\mu & 0 & 0 \\ -n\lambda & s+(n-1)\lambda+\mu & -\mu & 0 \\ 0 & (n-1)\lambda & s+(n-2)\lambda+\mu & 0 \\ 0 & 0 & (n-2)\lambda & s \end{bmatrix}^{-1}. \tag{19}$$

We are interested in P_{n-3}. One can study the roots of the inverse s matrix to find that P_{n-3} has a pole at $s=0$ with coefficient 1 and another one with coefficient 1 at $s=S_1$ where

$$S_1 \simeq \lambda n(n-1)(n-2)\left(\frac{\lambda}{\mu}\right)^2 \tag{20}$$

Inverse Laplace transforming, we can calculate the reliability as

$$\begin{aligned} \mathrm{R}(t) &= 1-p_{n-1}(t) \simeq 1-(1-e^{-t/\ \mathrm{MTTDL}_2}) = e^{-t/\ \mathrm{MTTDL}_2}, \\ \mathrm{MTTDL}_2 &= \frac{1}{n(n-1)(n-2)\lambda}\left(\frac{\mu}{\lambda}\right)^2. \end{aligned} \tag{21}$$

c **redundancies**

Recognizing the patterns in Eqs. 13, 16 and 17 we can write the reliability formula for c-redundant system

$$\begin{aligned} \mathrm{R}(t) &\sim e^{-t/\mathrm{MTTDL}_c}, \\ \mathrm{MTTDL}_c &= \left(\frac{\mu}{\lambda}\right)^c \frac{(n-c-1)!}{\lambda n!}, \end{aligned} \tag{22}$$

Eq. 22 is what we will use to calculate the reliability in the following sections. The most important observation here is that the system is defined by a reliability function which is that of an exponential distribution! Therefore we can use the properties of exponential distribution to calculate other metrics such as availability, which will be done in the following sections.

Parallel repairs

We assumed that repair rate is constant no matter how many failures you have. In reality restoration may be going on in parallel, so if you have two failures to restore and you work on them in parallel, the repair rate is effectively doubled, as illustrated in Figure 8.

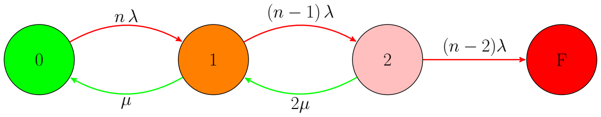


Figure 8: Markov chain for a system with two redundancies

In this case the μ entries in the s matrices need to be scaled appropriately. Following the same steps one gets

$$\begin{aligned} \mathrm{R}(t) &\sim e^{-t/\ \mathrm{MTTDL}_{c(p)}}, \\ \mathrm{MTTDL}_{c(p)} &= \left(\frac{\mu}{\lambda}\right)^{c} \frac{c!(n-c-1)!}{\lambda n!}, \end{aligned} \tag{23}$$

which differs from Eq. 22 only by the $c!$ factor, and this makes a difference only when $c > 1$, as expected.

Hard Errors

Now we want to include a failure mode that we have ignored so far: unrecoverable errors (UER), i.e., hard errors. The tricky part about hard errors is that you will not know if a sector is corrupted until you attempt to read the very sector. Some systems do scrubbing of drives. When the system is idle, it reads the entire disk to locate and fix UERs. Obviously this causes performance loss and one can only afford it at a limited frequency[7]. If a sector is corrupted, then you can recover it from its parities stored. However, one can easily see this will cause immediate data loss if the system is in the critical mode. For example a system with c redundancy is in critical mode when it has c failures, that is, it is trying to recover c shards of data by reading from all of the remaining shards. If a UER is encountered at this point, there is no parity left to reconstruct the data.

Let's start with the simplest case: a system with one redundancy, as in Figure 9.

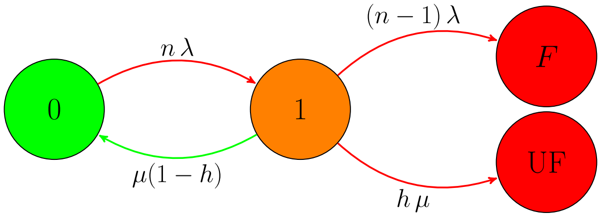


Figure 9: Markov chain for a system with one redundancy. Hard errors cause failures in the critical mode.

h is the probability of getting a UER during the rebuild of a failure, and it enters the link with a coefficient of μ since μ sets the data reading rate. i.e μh is the rate of observing UERs when $n-1$ drives are read. If UER is observed during the rebuild, it will be system failure. Therefore, the recovery rate reduces to $(1-h)\mu$ whereas the transition rate from $n-1$ to $n-2$ state increases to $(n-1)\lambda+h\mu$. It is important to explicitly define what h is. To reconstruct a failed drive of capacity C_{bits}, the system will need to read all the data in all $N-1$ healthy drives and push the data through Reed Solomon inversion calculations to reconstruct the data in the failed drive. So the total amount of data to be read is $(n-1)C_{\text{bits}}$. And h is the probability of getting one or more UERs for the entire data, which can be calculated as follows:

$$h = 1-(1-\ \text{UER})^{(n-1)C_{\text{bits}}} \simeq 1-e^{-UER(n-1)C_{\text{bits}}}. \tag{24}$$

A trick from CALC101

Did you notice the difficulty in the expression $(1-\ \text{UER})^{(n-1)C_{\text{bits}}}$ in Eq. 24? UER is a tiny number $\propto 10^{-15}$. $(n-1)C_{\text{bits}}$ is a huge number $\propto 10^{14}$. If the expression is computed numerically, it may just exceed the precision of the computation. Let's do ourselves a favor, and use a trick taught in the early days of first calculus classes. Consider the following limit:

$$\lim_{N\to\infty}\left(1-\frac{x}{N}\right)^N = \lim_{N\to\infty}e^{N\ln(1-\frac{x}{N})} = \lim_{N\to\infty}e^{N(-\frac{x}{N}+\mathcal{O}(1/N^2))} = e^{-x}, \tag{25}$$

which we used in Eq. 24. Note that now we have the multiplication of a very small number and a very large one, $UER(n-1)C_{\text{bits}}$, which will result in a numerical value that will not require high precision in the calculations.

Equation 24 shows that h is a function of the amount of drives to be read, i.e. $n-1$, and we will use h_{n-1} to emphasize this fact.

The S-matrix for the system becomes

$$\begin{bmatrix} s+n\lambda & -\mu(1-h_{n-1}) & 0 \\ -n\lambda & s+(n-1)\lambda+\mu & 0 \\ 0 & -(n-1)\lambda-h_{n-1}\mu & s \end{bmatrix}, \tag{26}$$

determinant of which will have 3 roots. We will be interested in smallest non zero root:

$$\begin{aligned} s_3 &= \frac{\mu+(2n-1)\lambda}{2}+\frac{1}{2}\sqrt{\lambda^2-2\lambda\mu+\mu^2+4\lambda\mu n-4h_{n-1}\lambda\mu n} \\ &\simeq \frac{n\lambda[(n-1)\lambda+\mu h_{n-1}]}{\mu+(2n-1)\lambda} \simeq \frac{n\lambda[(n-1)\lambda+\mu h_{n-1}]}{\mu}, \end{aligned} \tag{27}$$

where we assumed $(2n-1)\lambda \ll \mu$. s_3 is what will go into the exponent once we inverse Laplace transform. The corresponding MTTF becomes

$$\text{MTTF} = \frac{1}{s_3} \simeq \frac{\mu}{n\lambda[(n-1)\lambda+\mu h_{n-1}]}. \tag{28}$$

And as expected, this reduces to Eq. 16 when $h_{n-1}=0$. It is illustrative to break out the MTTF into two modes: the drive failure mode (DF) and the mode related to UER. This decomposition allows us to write the total MTTF as follows:

$$\begin{aligned} \frac{1}{\text{MTTF}} &= \frac{1}{\text{MTTDL}_{\text{ DF}}}+\frac{1}{1/(n\lambda)}h_{n-1}, \\ \text{MTTDL}_{\text{ DF}} &= \frac{\mu}{n\lambda[(n-1)\lambda]}, \end{aligned} \tag{29}$$

and we will recognize the factor $1/(n\lambda)$ as the MTTDL of the system with no redundancy. And in fact this is a general pattern:

$$\frac{1}{\text{MTTF}_c} = \frac{1}{\text{MTTDL}_{c,\text{ DF}}}+\frac{h_{n-1}}{\text{MTTDL}_{c-1,\text{ DF}}}, \tag{30}$$

where MTTDL_c is given in Eq. 22. To get a feel for h_{n-1}, let's plug in typical numbers: $\text{UER}=10^{-15}$, $n=20$ and $C=10$TB.

$$h_{n-1} = 1-(1-10^{-15})^{19*8*10^{13}} = 1-e^{19*8*10^{13}*10^{-15}} = 0.78, \tag{31}$$

which is again the probability of having at least one bad sectors when all $n-1$ drives are read to reconstruct the failed one(s). This tells us that with significant probability, we are going to lose data if the system ever comes down to the critical mode.

If h_{n-1} is not small enough, MTTF_c will be dominated by the second term in Eq. 30

$$\text{MTTF}_c \simeq \frac{\text{MTTDL}_{c-1,\text{ DF}}}{h_{n-1}}, \tag{32}$$

which highlights the fact that redundancy is effectively reduced by one unit due to UERs.

Simulation

I have written a couple of Monte Carlo codes to simulate the durability. You can find them in my repository or copy them below. If you want to play with the code without having to fork it or copy it to run on your local computer, I also pulled the Python code into the colab of Google as a Jupyter notebook. You can modify/run it on your browser here. The codes below are essentially identical other than the fact that they are in different languages. The following two codes run Monte Carlo simulation without UREs.

Figure 10 shows the simulation results for various input parameters(see the caption) and at various times of the system.

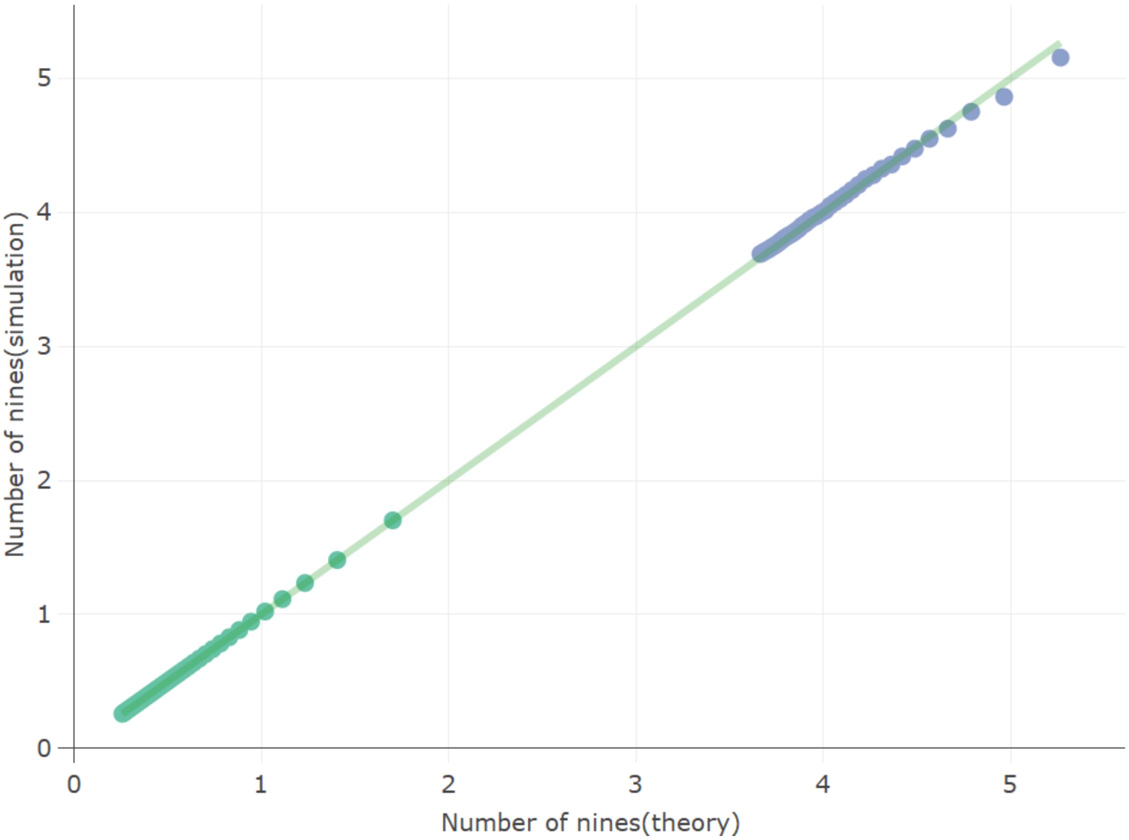


Figure 10: The comparison of theory vs Monte Carlo results with various input parameters. The results show that Markov chain model is very successful.

```
$[n, c, \text{AFR}, \text{UER}, \text{drive cap}, \text{recovery rate}] \leftarrow \text{inputs}$
$\text{UER prob} \leftarrow h(UER)$
$\text{Simulation size} \leftarrow \text{Estimated MTTDL}$
```

```
Loss Count ← 0
Sector Loss Count ← 0
for sys < Simulation size do
    to be repaired ← [ ]
    Failure times ← $n$ × Exponential Random Number
    Failure times ← Failure times[Failure times < $1\,year$]
    for Failure time in Failure times do
        add(Failure time,to be repaired)
        to be repaired ← to be repaired[to be repaired>time now- repair time]
        if length(to be repaired ) $> c$ then
            Loss Count++
            Sector Loss Count++
        if length(to be repaired ) $= c$ then
            if rand uniform$(0, 1) <$ UER prob then
                Sector Loss Count++
return [Loss Count,Sector Loss Count]
```

Battle of models

We have 3 contenders: Wasabi, BackBlaze and Markov. Which one is the most accurate, and how do you even judge? I will use the simulation results as the golden standard. And, as data durability means the probability of not losing any data, I will use the durability with URE as the deciding metric. The simulation code above reports durability with and without UREs, so we can still compare the durability numbers without URE, however it is not the most appropriate one. One would be still upset to lose a sector of data since it might corrupt a much larger data set.

Furthermore, I will run the simulation with fewer parities and a bit higher failure rates so that simulation captures enough data loss events to estimate the probabilities. For higher redundancies, the simulation will take too long to complete (In order to make the script work at higher redundancies, I added an argument to disable the simulation and report the numbers based on Markov chain model, that is, it can be used as a calculator by skipping the simulation.)

Below is the output for a sample run:

Running 40000000 simulations: 20 total shards with 2 parities, i.e.,18+2, 1.0% AFR, uer=1.0 * 10^-{15}, 20.0TB drive capacity, and 50.0MB/s recovery speed (4.6 days).

prob of UER=0.94. Theoretical predictions: NoN= 6.25 nines, NoN wUER= 3.34 nines.

Simulation Results:21 data loss instances: NoN= 6.28 nines, NoN wUER= 3.34 nines which shows a remarkable agreement between the simulation and the Markov chain model. The true durability number is the one with URE, that is 3.3 nines. UREs ignored, you get 6.3 nines.

Compare this against BackBlaze number, 7 nines, which you can compute yourself by going back to BackBlaze model and adjusting the inputs. This shows that BackBlaze Model is overestimating the durability without UREs at these inputs by 0.7 nines, which is a factor of 5. The bigger issue is that the real durability must include UREs and it is 3.3 nines.

And compare this against Wasabi number, 12 nines, which you can compute yourself by going back to Wasabi model and adjusting the inputs. This shows that Wasabi Model is not so good!

Conclusions

Calculating data durability is a nontrivial business and requires great care in the math. It is always a good idea to verify models with simulations, and that is typically what people fail to do. The detailed mathematical analysis and simulations show that Markov chain based model predicts the data durability very accurately, whereas some other models are way off.

It is also important to note that any model is as good as its assumptions, a.k.a. garbage in garbage out. The most important assumption we made was that failures can be described by an exponential distribution. That may not be the valid for certain cases. For example, if the drives are failing at higher rates as the population ages, a Weibull distribution might be a better fit. Markov chain analysis can be extended to address such cases.

[1] W. C. Storage, "Wasabi tech brief 2019," 2019 [Online]. Available: https://wasabi.com/wp-content/uploads/2019/10/Durability-Tech-Brief-2019.pdf

[2] B. Wilson, "Cloud storage durability," 2018 [Online]. Available: https://www.backblaze.com/blog/cloud-storage-durability/

[3] B. Beach, "Python code to calculate the durability of data stored with erasure coding," 2018 [Online]. Available: https://github.com/Backblaze/erasure-coding-durability/

[4] W. A. Burkhard and J. Menon, "MDS disk array reliability," *Technical Report CS92-269*, Nov. 1993 [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.53.2063&rep=rep1&type=pdf

[5] A. Papoulis and U. Pillai, *Probability, random variables and stochastic processes*, 4th ed. McGraw-Hill, 2001.

[6] Wikipedia, "Markov chain — Wikipedia, the free encyclopedia." http://en.wikipedia.org/w/index.php?title=Markov%20chain&oldid=901189015, 2019.

[7] I. Iliadis, R. Haas, X.-Y. Hu, and E. Eleftheriou, "Disk scrubbing versus intra-disk redundancy for high-reliability raid storage systems," vol. 36, no. 1, pp. 241–252, Jun. 2008, doi: 10.1145/1384529.1375485. [Online]. Available: https://doi.org/10.1145/1384529.1375485