ZFS STORAGE POOL LAYOUT

Storage and Servers Driven by Open Source

marketing@iXsystems.com

CONTENTS

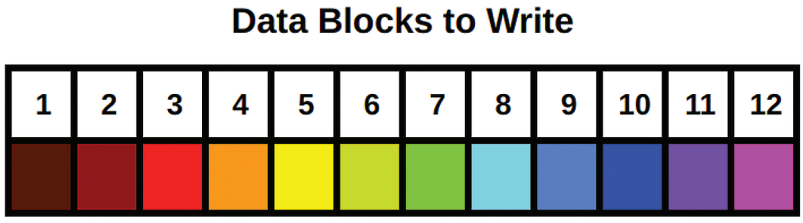
1 INTRODUCTION AND EXECUTIVE SUMMARY

The layout of a ZFS storage pool has a significant impact on system performance under various workloads. Given the importance of picking the right configuration for your workload, and the fact that making changes to an in-use ZFS pool is far from trivial, it is important for an administrator to understand the mechanics of pool performance when designing a storage system.

To quantify pool performance, we will consider six primary metrics:

- Read I/O operations per second (IOPS)
- Write IOPS
- Streaming read speed
- Streaming write speed
- Storage space efficiency (usable capacity after parity versus total raw capacity)
- Fault tolerance (maximum number of drives that can fail before data loss)

For the sake of comparison, we'll use an example system with 12 drives, each one sized at 6TB, and say that each drive does 100MB/s streaming reads and writes and can do 250 read and write IOPS. We will visualize how the data is spread across the drives by writing 12 multi-colored blocks, shown below. The blocks are written to the pool starting with the brown block on the left (number one), and working our way to the pink block on the right (number 12).



Note that when we calculate data rates and IOPS values for the example system, they are only approximations. Many other factors can impact pool access speeds for better (compression, caching) or worse (poor CPU performance, not enough memory).

There is no single configuration that maximizes all six metrics. Like so many things in life, our objective is to find an appropriate balance of the metrics to match a target workload. For example, a cold-storage backup system will likely want a pool configuration that emphasizes usable storage space and fault tolerance over the other data-rate focused metrics.

Let's start with a quick review of ZFS storage pools before diving into specific configuration options. ZFS storage pools are comprised of one or more virtual devices, or vdevs. Each vdev is comprised of one or more storage providers, typically physical hard disks. All disk-level redundancy is configured at the vdev level. That is, the RAID layout is set on each vdev as opposed to on the storage pool. Data written to the storage pool is then striped across all the vdevs. Because pool data is striped across the vdevs, the loss of any one vdev means total pool failure. This

is perhaps the single most important fact to keep in mind when designing a ZFS storage system. We will circle back to this point in the sections below, but keep it in mind as we go through the vdev configuration options.

Because storage pools are made up of one or more vdevs with the pool data striped over top, we'll take a look at pool configuration in terms of various vdev configurations. There are three basic vdev configurations: striping, mirroring, and RAIDZ (which itself has three different varieties).

2 STRIPED VDEV

A striped vdev is the simplest configuration. Each vdev consists of a single disk with no redundancy. When several of these single-disk, striped vdevs are combined a single storage pool, the total usable storage space would be the sum of all the drives. When you write data to a pool made of striped vdevs, the data is broken into small chunks called "blocks" and distributed across all the disks in the pool. The blocks are written in "round-robin" sequence, meaning after all the disks receive one row of blocks, called a *stripe*, it loops back around and writes another stripe under the first. A striped pool has excellent performance and storage space efficiency, but ***absolutely zero fault tolerance.*** If even a single drive in the pool fails, the entire pool will fail and all data stored on that pool will be lost.

The excellent performance of a striped pool comes from the fact that all of the disks can work independently for all read and write operations. If you have a bunch of small read or write operations (IOPS), each disk can work independently to fetch the next block. For streaming reads and writes, each disk can fetch the next block in line synchronized with its neighbors. For example, if a given disk is fetching block *n*, its neighbor to the left can be fetching block *n*-1, and its neighbor to the right can be fetching block *n*+1. Therefore, the speed of all read and write operations as well as the quantity of read and write operations (IOPS) on a striped pool will scale with the number of vdevs. Note here that I said the speeds and IOPS scale with number of vdevs rather than number of drives; there's a reason for this and we'll cover it below when we discuss RAID-Z.

Here's a summary of the total pool performance (where N is the number of disks in the pool):

_N_-wide striped:

- Read IOPS: N * Read IOPS of single drive
- Write IOPS: N * Write IOPS of single drive
- Streaming read speed: N * Streaming read speed of single drive
- Streaming write speed: N * Streaming write speed of single drive
- Storage space efficiency: 100%
- Fault tolerance: **None!**

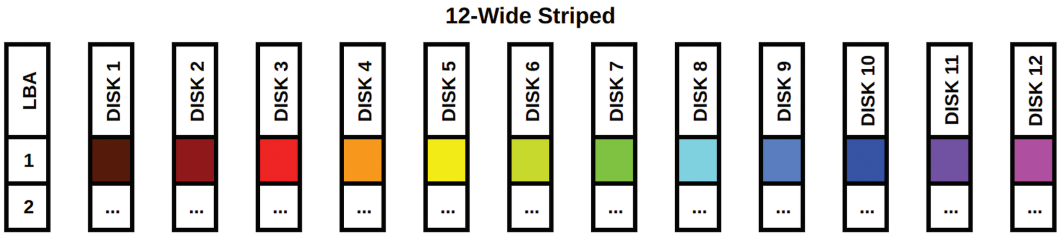
Let's apply this to our example system, configured with a 12-wide striped pool:

12-wide striped:

- Read IOPS: 3000
- Write IOPS: 3000
- Streaming read speed: 1200 MB/s
- Streaming write speed: 1200 MB/s

- Storage space efficiency: 72 TB
- Fault tolerance: **None!**

Below is a visual depiction of our 12 rainbow blocks written to this pool configuration:



The blocks are simply striped across the 12 disks in the pool. The LBA column on the left stands for "Logical Block Address". If we treat each disk as a column in an array, each LBA would be a row. It's also easy to see that if any single disk fails, we would be missing a color in the rainbow and our data would be incomplete. While this configuration has fantastic read and write speeds and can handle a ton of IOPS, the data stored on the pool is very vulnerable. This configuration is not recommended unless you're comfortable losing all your pool's data whenever any single drive fails.

3 MIRRORED VDEV

A mirrored vdev consists of two or more disks. A mirrored vdev stores an exact copy of all the data written to it on each one of its drives. Traditional RAID-1 mirrors usually only support two drive mirrors, but ZFS allows for more drives per mirror to increase redundancy and fault tolerance. All disks in a mirrored vdev have to fail for the vdev, and thus the whole pool, to fail. Total storage space will be equal to the size of a single drive in the vdev. If you're using mismatched drive sizes in your mirrors, the total size will be that of the smallest drive in the mirror.

Streaming read speeds and read IOPS on a mirrored vdev will be faster than write speeds and IOPS. When reading from a mirrored vdev, the drives can "divide and conquer" the operations, similar to what we saw above in the striped pool. This is because each drive in the mirror has an identical copy of the data. For write operations, all of the drives need to write a copy of the data, so the mirrored vdev will be limited to the streaming write speed and IOPS of a single disk.

Here's a summary:

***N*-way mirror:**

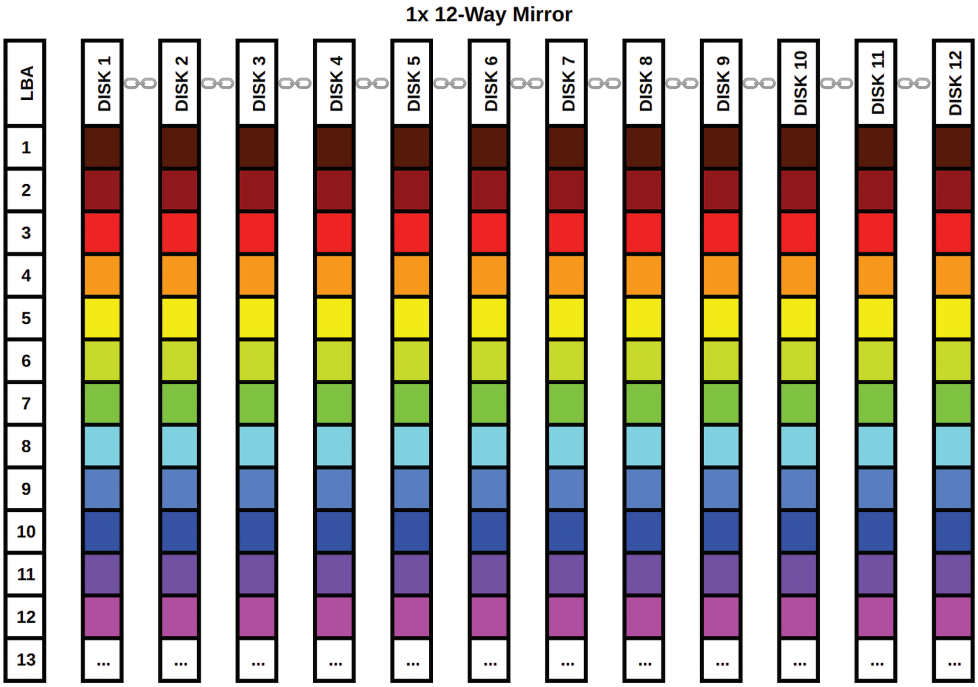
- Read IOPS: *N* * Read IOPS of single drive
- Write IOPS: Write IOPS of single drive
- Streaming read speed: *N* * Streaming read speed of single drive
- Streaming write speed: Streaming write speed of single drive
- Storage space efficiency: 50% for 2-way, 33% for 3-way, 25% for 4-way, etc. [*(N-1)/N*]
- Fault tolerance: 1 disk per vdev for 2-way, 2 for 3-way, 3 for 4-way, etc. [*N-1*]

For our first example configuration, let's do something ridiculous and create a 12-way mirror. ZFS supports this kind of thing, but your management probably will not.

1x 12-way mirror:

- Read IOPS: 3000
- Write IOPS: 250
- Streaming read speed: 1200 MB/s
- Streaming write speed: 100 MB/s
- Storage space efficiency: 8.3% (6 TB)
- Fault tolerance: 11

Let's look at this configuration visually:



As we can clearly see from the diagram, every single disk in the vdev gets a full copy of our rainbow data. The chainlink icons between the disk labels in the column headers indicate the disks are part of a single vdev. We can lose up to 11 disks in this vdev and still have a complete rainbow. Of course, the data takes up far too much room on the pool, occupying a full 12 LBAs in the data array.

Obviously, this is far from the best use of 12 drives. Let's do something a little more practical and configure the pool with the ZFS equivalent of RAID-10. We'll configure six 2-way mirror vdevs. ZFS will stripe the data across all 6 of the vdevs. We can use the work we did in the striped vdev section to determine how the pool as a whole will behave. Let's first calculate the performance per vdev, then we can work on the full pool:

1x 2-way mirror:

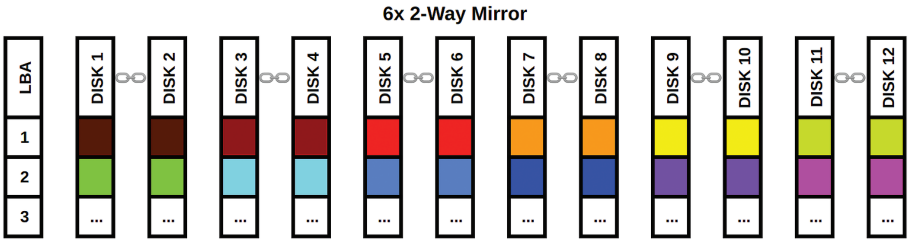
- Read IOPS: 500
- Write IOPS: 250
- Streaming read speed: 200 MB/s
- Streaming write speed: 100 MB/s
- Storage space efficiency: 50% (6 TB)
- Fault tolerance: 1

Now we can pretend we have 6 drives with the performance statistics listed above and run them through our striped vdev performance calculator to get the total pool's performance:

6x 2-way mirror:

- Read IOPS: 3000
- Write IOPS: 1500
- Streaming read speed: 3000 MB/s
- Streaming write speed: 1500 MB/s
- Storage space efficiency: 50% (36 TB)
- Fault tolerance: 1 per vdev, 6 total

Again, we will examine the configuration from a visual perspective:

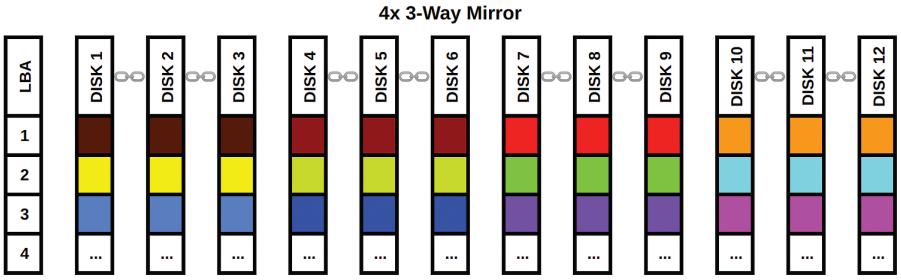


Each vdev gets a block of data and ZFS writes that data to all of (or in this case, both of) the disks in the mirror. As long as we have at least one functional disk in each vdev, we can retrieve our rainbow. As before, the chain link icons denote the disks are part of a single vdev. This configuration emphasizes performance over raw capacity, but doesn't totally disregard fault tolerance as our striped pool did. It's a very popular configuration for systems that need a lot of fast I/O. Let's look at one more example configuration using four 3-way mirrors. We'll skip the individual vdev performance calculation and go straight to the full pool:

4x 3-way mirror:

- Read IOPS: 3000
- Write IOPS: 1000
- Streaming read speed: 3000 MB/s

- Streaming write speed: 400 MB/s
- Storage space efficiency: 33% (24 TB)
- Fault tolerance: 2 per vdev, 8 total



While we have sacrificed some write performance and capacity, the pool is now extremely fault tolerant. This configuration is probably not practical for most applications and it would make more sense to use lower fault tolerance and set up an offsite backup system.

Striped and mirrored vdevs are fantastic for access speed performance, but they impose at least a 50% penalty on the total usable space of your pool. With RAIDZ, we can keep data redundancy without sacrificing as much storage space efficiency.

4 RAIDZ VDEV

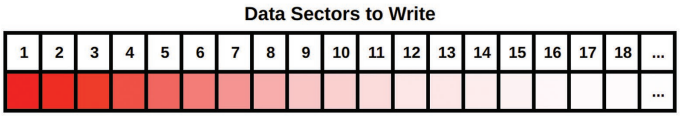
RAIDZ is comparable to traditional RAID-5 and RAID-6. RAIDZ comes in three flavors: RAIDZ1, Z2, and Z3, where Z1 uses single parity, Z2 uses double parity, and Z3 uses triple parity. When data is written to a RAIDZ vdev, it is striped across the disks but ZFS adds in parity information. This means we have a little bit more stuff to store on the disk, but in return, we can recover from a certain number of drive failures in the vdev. The parity information on each stripe is computed from the data written to that stripe. If a drive fails, we can reverse the formula of that computation in order to recover the missing data. RAIDZ1 adds one sector of parity data per stripe and can recover from a single drive failure per vdev. RAIDZ2 and Z3 add two and three sectors per stripe, and can recover from two and three drive failures per vdev, respectively.

For RAIDZ performance, the terms *parity disks* and *data disks* refer to the parity level (1 for Z1, 2 for Z2, and 3 for Z3; we'll call the parity level p) and vdev width (the number of disks in the vdev, which we'll call *N*) minus *p*. The effective storage space in a RAIDZ vdev is equal to the capacity of a single disk times the number of data disks in the vdev. If you're using mismatched disk sizes, it's the size of the smallest disk times the number of data disks. Fault tolerance per vdev is equal to the parity level of that vdev.

Measuring I/O performance on RAIDZ is a bit trickier than our previous examples. ZFS breaks write data into pieces called blocks and stripes them across the vdevs. Each vdev breaks those blocks into even smaller chunks called sectors. For striped vdevs, the sectors are simply written sequentially to the drive. For mirrored vdevs, all sectors are written sequentially to each disk. On RAIDZ vdevs however, ZFS has to add additional sectors for the parity information. When a RAIDZ vdev gets a block to write out, it will divide that block into sectors, compute all the parity information, and hand each disk either a set of data sectors or a set of parity sectors. ZFS ensures that there are *p* parity sectors for each stripe written to the RAIDZ vdev.

I/O operations on a RAIDZ vdev need to work with a full block, so each disk in the vdev needs to be synchronized and operating on the sectors that make up that block. No other operation can take place on that vdev until all the disks have finished reading from or writing to those sectors. Thus, IOPS on a RAIDZ vdev will be that of a single disk. While the number of IOPS is limited, the streaming speeds (both read and write) will scale with the number of data disks. Each disk needs to be synchronized in its operations, but each disk is still reading/writing unique data and will thus add to the streaming speeds, minus the parity level as reading/writing this data doesn't add anything new to the data stream.

Because a RAIDZ vdev splits individual blocks into sector-sized chunks, our rainbow-colored blocks example needs some tweaking. Each individual color needs to be broken up into sectors. To represent the division of a single block into multiple sectors, we'll use a single-color gradient, an example of which is shown below:



This single data block is shown as continuing on past its 18th sector with the ellipsis at the end of the block. We have represented it this way because ZFS uses variable block sizes when writing data to vdevs. This has important implications in ZFS deployments, particularly for RAIDZ configurations. F or now, let's look at general RAIDZ performance. Here's a summary:

***N*-wide RAIDZ, parity level *p*:**

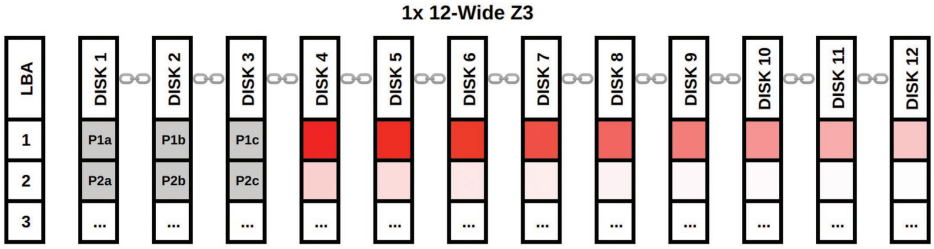
- Read IOPS: Read IOPS of single drive
- Write IOPS: Write IOPS of single drive
- Streaming read speed: (*N* - *p*) * Streaming read speed of single drive
- Streaming write speed: (*N* - *p*) * Streaming write speed of single drive
- Storage space efficiency: (*N* - *p*)/*N*
- Fault tolerance: 1 disk per vdev for Z1, 2 for Z2, 3 for Z3 [*p*]

We'll look at three example RAIDZ configurations. The first will use a single vdev: a 12-wide Z3 array.

1x 12-wide Z3:

- Read IOPS: 250
- Write IOPS: 250
- Streaming read speed: 900 MB/s
- Streaming write speed: 900 MB/s
- Storage space efficiency: 75% (54 TB)
- Fault tolerance: 3

Based on these numbers, this looks like it could be a decent option unless you need to handle lots of IOPS. Below is a visual depiction of a single block of data being written to a pool with this configuration. The data sectors are colored in shades of red and the parity sectors are grey.



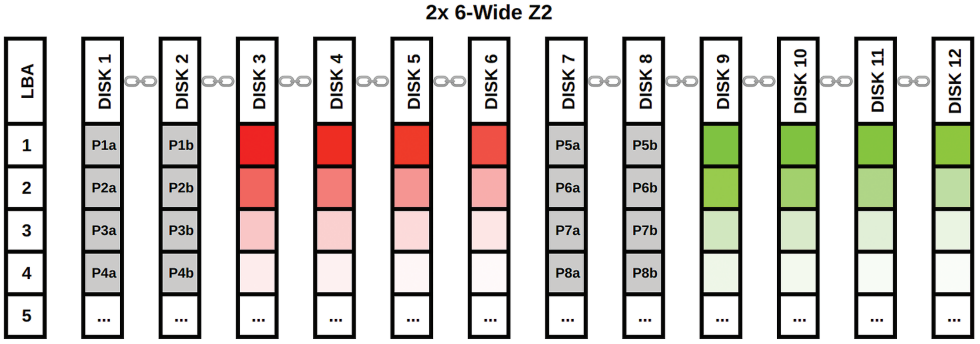
In this diagram, we can see that each stripe of data on the vdev gets its own set of parity sectors. Each of these parity sectors are unique, even on a given stripe, which is why they are labeled “P1a”, “P1b”, etc. If each parity sector in a given stripe was identical, having multiple copies would not provide us any more information than having a single copy of that parity sector! In that case, we wouldn’t have enough information to recover data after multiple drive failures. With this RAIDZ3 configuration, we can lose three of the disks with data sectors on them and use the parity information to recover the data from those dead drives. If we lose drives with parity sectors, we can simply recompute the missing parity data.

Now let’s look at configuring two vdevs, each a 6-wide Z2 array. I’ll skip the single vdev stats and jump right to the full pool stats:

2x 6-wide Z2:

- Read IOPS: 500
- Write IOPS: 500
- Streaming read speed: 800 MB/s
- Streaming write speed: 800 MB/s
- Storage space efficiency: 66.7% (48 TB)
- Fault tolerance: 2 per vdev, 4 total

This configuration sacrifices a bit of streaming speed and some capacity to double the IOPS. To visualize this configuration, we will write two blocks of data to the pool. Each Z2 vdev will get a single block that gets split into sectors. As above, the data sectors are shades of red and green, and the parity sectors are grey.



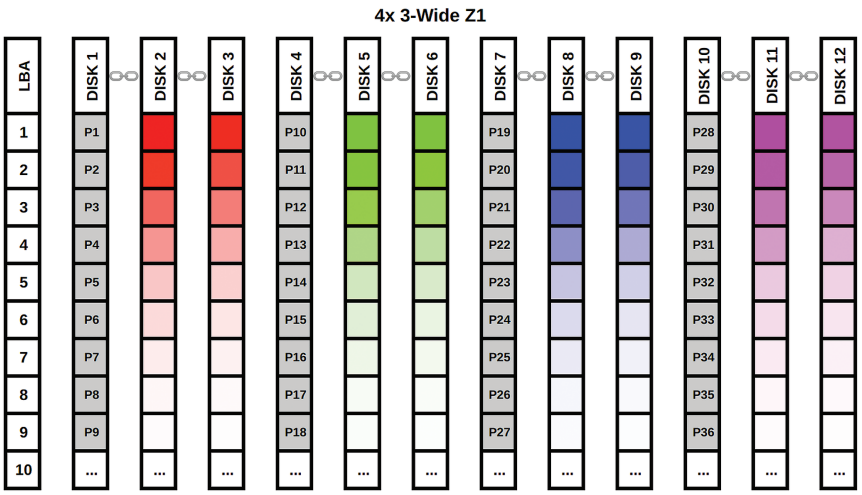
As in the Z3 diagram, each data stripe gets its own pair of unique parity sectors. The first data block is written to the first vdev and the second data block is written to the second vdev. A third data block would again be written to the first vdev, and so on.

The last configuration uses four vdevs, each a 3-wide Z1 array.

4x 3-wide Z1:

- Read IOPS: 1000
- Write IOPS: 1000
- Streaming read speed: 800 MB/s
- Streaming write speed: 800 MB/s
- Storage space efficiency: 66.7% (48 TB)
- Fault tolerance: 1 per vdev, 4 total

This configuration sacrifices some of its fault tolerance to double the IOPS. For this diagram, we'll write four blocks of data. Again, each vdev will get a single block and split it into sectors.



Each stripe gets its own parity sector, but unlike the previous examples, we only have a single parity sector per data stripe. This is why RAIDZ1 is not highly fault tolerant and is thus not a recommended configuration for storing mission-critical data.

I want to make a few quick points on fault tolerance and pool failure probability before we move on. If a single vdev in a pool is lost, your data is lost. The configurations we discussed above all use pools made up of identical vdevs. Using identical vdevs is strongly recommended, but it is possible to mismatch vdevs in a pool. For example, you could configure an 11-wide Z3 vdev and add a single striped vdev as the 12th drive in the pool. This would not be smart. Your extremely fault-tolerant Z3 vdev now depends on that single 12th drive to maintain your data. If that drive goes, your whole pool is gone.

The question of translating per-vdev fault tolerance into total pool reliability is more complicated than it might

initially appear. For example, a 12-wide Z3 pool with 3 parity drives is statistically less likely to fail than a 2x 6-wide Z2 pool with 4 total parity drives. Our 6x 2-way mirror pool has 6 total parity drives, but it's far more likely to fail than either the Z3 or Z2 configurations. The 4x 3-wide Z1 configuration has an even higher failure probability than the mirrors. The moral is, don't simply look at the total number of parity drives and think "more is better". If you're interested in the math behind pool failure probability, check out the post here: http://jro.io/r2c2/.

5 EXAMPLES BY WORKLOAD

We now have some rough numbers to quantify pool performance based on its configuration, but how do we translate that to real-world applications? This can often be the more difficult part of the ZFS pool configuration question because it requires an accurate understanding of the workload. Let's take a look at a few example scenarios and decide which configuration would be the best fit for that given workload.

Scenario 1: Data backup system and low-access file share

We want to configure a ZFS storage system to house automated data backups and to function as a file share for a small handful of users. Under this workload, IOPS are likely not as important as streaming speeds. We'll also want good storage efficiency and good fault tolerance. Assuming the same example 12-drive system, we might go with either the 2x 6-wide RAIDZ2 configuration or the 1x 12-wide RAIDZ3 setup. We can decide between these two configurations based on how many users will be accessing the system simultaneously (how many IOPS can we expect). If our backups hit the system at midnight and during business hours we only have two or three people connected to the file share, we can probably get away with the lower IOPS Z3 configuration. If we have more users in the system or we have backups hitting during business hours, it may be worth sacrificing some capacity to get higher IOPS with the Z2 configuration.

Scenario 2: iSCSI host for database VM storage

We have several database VMs that will be using our system for storage. We'll serve up the storage with iSCSI and we need the data to move as quickly as possible. The databases will be regularly backed up, so we aren't terribly concerned with data loss, but we don't want a drive failure to halt all VM operations while we restore from backup. The more VMs we are hosting, the more IOPS the system will have to handle. The obvious choice here is a set of mirrored vdevs. The more mirrors we have in the system, the more performance we can expect. Even if a drive in the system fails, we can recover quickly and with no down time by swapping the drive and resilvering the mirror. If we tried to use a Z2 or Z3 configuration to get some more storage space from the system, VM performance would likely be poor due to low pool IOPS.

Scenario 3: High-resolution video production work via file share

We have a group of video editors that need to work on high-resolution footage stored on our system. They will be editing the footage directly from the pool, as opposed to copying it to local storage first. Streaming speeds will be very important as high resolution video files can have gigantic bitrates. The more editors we have, the more performance we'll need. If we only have a small handful of editors, we can probably get away with several RAIDZ2 vdevs, but as you add more editors, IOPS will become increasingly important to support all their simultaneous IO work. At a certain point, Z2 will no longer be worth its added capacity and a set of mirrored vdevs will make more sense. That exact cutoff point will vary, but will likely be between 5 and 10 total editors working simultaneously.

There are two special vdev types that we have not discussed: an L2ARC and a SLOG. These special vdevs can be added to a pool to function as a read cache and a write cache, respectively. Typically, you would use an SSD for these vdevs. You should consider adding an L2ARC if your workload demands high read IOPS and a SLOG if your workload demands high write IOPS. If you're considering deploying a system with an L2ARC or a SLOG, I would encourage you to contact a storage specialist at iXsystems.

ZFS storage pool configuration can certainly seem overwhelming, but that's because it offers so much flexibility to meet the needs of many different types of workloads. Indeed, many other aspects of ZFS follow this trend: its versatility can offer an enormous set of options and the simple task of determining the best option can seem daunting at first glance. Thankfully, once you dive into it, ZFS starts making sense fairly quickly. ZFS was originally created to make the lives of storage administrators easier, and once past the initial learning curve, it can do just that. Hopefully this guide has helped on that journey and you're well on your way to a successful ZFS deployment!