

[Storage](#) ▾[Servers](#) ▾[Solutions](#) ▾[Partners](#) ▾[Support](#) ▾[Blog](#)[Resources](#) ▾[Company](#) ▾

## Six Metrics for Measuring ZFS Pool Performance Part 2

Oct 2, 2018 | [Blog](#) | [5 comments](#)

In the first post, we discussed the importance of planning the ZFS pool layout which has a huge impact on how the system performs. To quantify this performance, we are looking at **six key metrics**:

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

For the sake of comparison, we are using an example system with 12 drives. Each drive has a capacity of 6TB, is capable of 100MB/s streaming reads and writes, and can do 250 read and write IOPS. Let's pick up where we left off and dive into RAID-Z.

### **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.

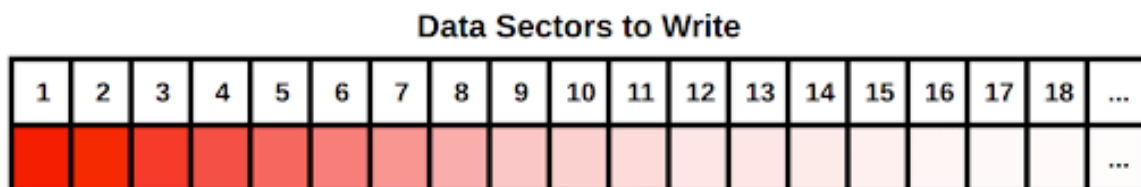
For RAIDZ performance, the terms *parity disks* and *data disks* refer to the parity level (1 for Z1, 2 for Z2, and 3 for Z3; we'll call the parity level  $p$ ) and vdev width (the number of disks in the vdev, which we'll call  $N$ ) minus  $p$ . The effective storage space in a RAIDZ vdev is equal to 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. For 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:  $(N - p) * \text{Streaming read speed of single drive}$
- Streaming write speed:  $(N - p) * \text{Streaming write speed of single drive}$
- Storage space efficiency:  $(N - p)/N$
- Fault tolerance: 1 disk per vdev for Z1, 2 for Z2, 3 for Z3 [*p*]

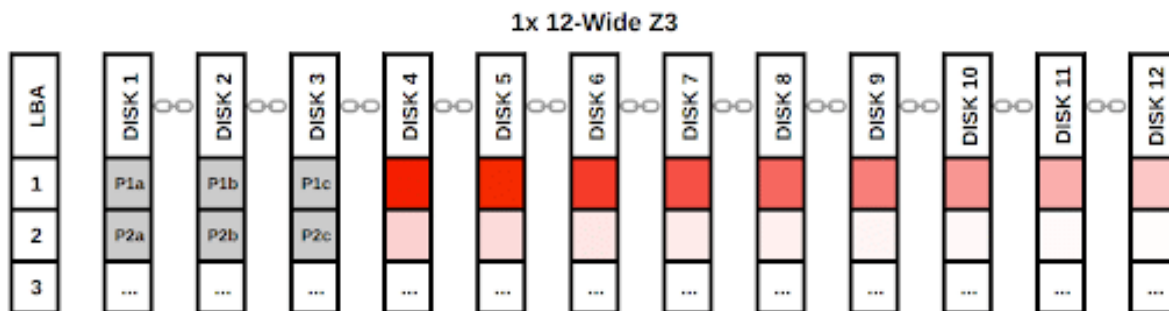
We'll look at three example RAIDZ configurations. The first will use a single vdev: a 12-wide Z3 array.

**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.

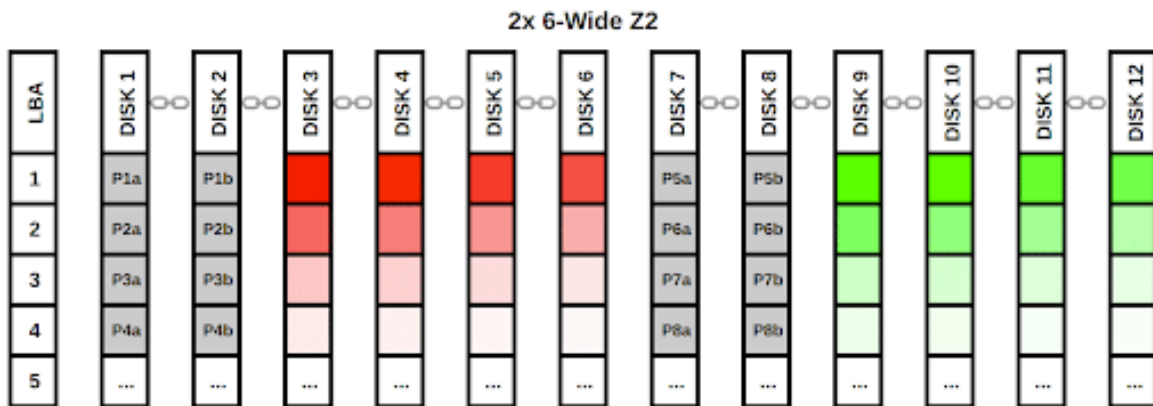
Now let’s look at configuring two vdevs, each a 6-wide Z2 array. I’ll skip the single vdev stats and jump right to the full pool stats:

### **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.



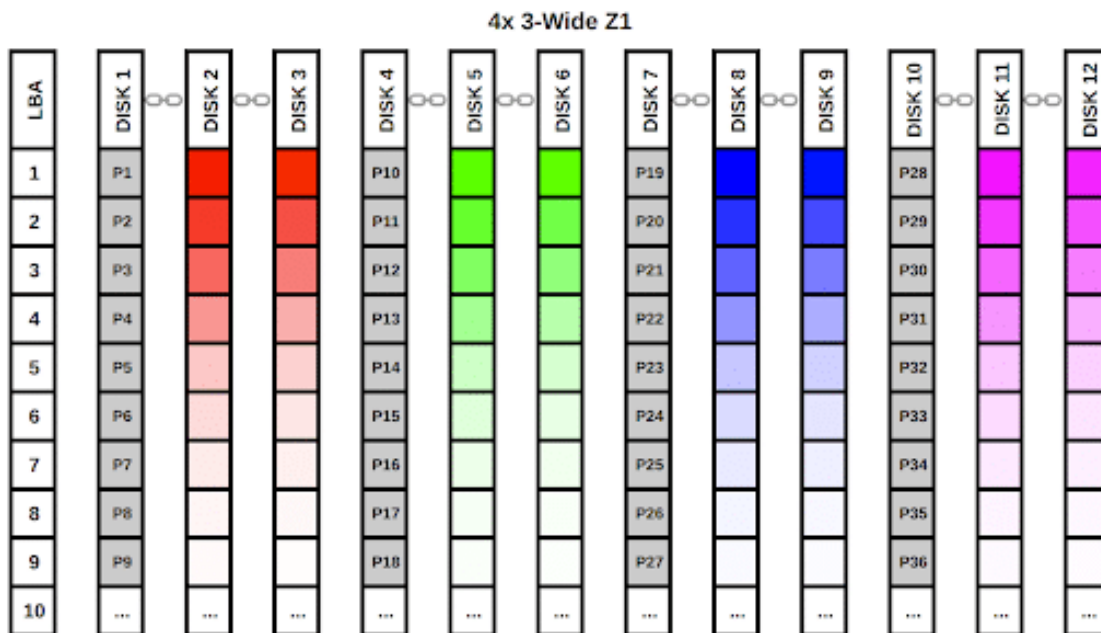
As in the Z3 diagram, each data stripe gets its own pair of unique parity sectors. The first data block is written to the first vdev and the second data block is written to the second vdev. A third data block would again be written to the first vdev, and so on.

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".

## **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 downtime 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 post and the [previous blog on ZFS performance](#) has helped on that journey and you're well on your way to a successful ZFS deployment!

Lastly, click here to view our [white paper](#) on measuring ZFS pool performance.

**Jason Rose, Sales Engineer**

## 5 Comments



**Emi** on October 2, 2018 at 9:27 pm

Nice write up, I especially liked the probability calculator part.

Thanks!

Reply



**Dirk Achenbach** on October 9, 2018 at 8:20 am

This is brilliant, many thanks.

Reply



**Joon Lee** on October 17, 2018 at 2:45 pm

Thank you for the support.

Reply



**Jeff** on October 12, 2018 at 5:05 pm

The performance comparisons are great; very useful. Consider a downloadable PDF link for your writeup?

Reply



**Joon Lee** on October 17, 2018 at 2:46 pm

Click on the “white paper” link up above or go here:  
[https://static.ixsystems.co/uploads/2018/10/ZFS\\_Storage\\_Pool\\_Layout\\_White\\_Paper\\_WEB.pdf](https://static.ixsystems.co/uploads/2018/10/ZFS_Storage_Pool_Layout_White_Paper_WEB.pdf)

Reply

## Follow Us

**13.3k** Follows



**Facebook** 3k Followers



**Twitter** 4.5k Followers



**Google+** 897 Followers



**YouTube** 2.4k Followers



**LinkedIn** 2.4k Followers

Recent

Popular

Tags



**LISA 2018 Recap**

November 2, 2018



**MeetBSD 2018: The Ultimate Hallway Track**

October 29, 2018



**Introducing the Asigra TrueNAS Backup Appliance**

October 23, 2018



## Ohio LinuxFest 2018 Recap

October 22, 2018



### Silicon Valley Veteran Morgan Littlewood Joins iXsystems as Senior Vice President, Product Management and Business...

October 9, 2018

[Next »](#)



# DOWNLOAD

Get your free Enterprise Storage Guide






# DOWNLOAD

Get your free Server Buying Guide

**The Ultimate Guide to  
Buying a New Server  
for Open Source**



 **systems**

11 Key Traits You **Must Absolutely**  
**DEMAND** From Your Provider



# SUBSCRIBE

Join our newsletter & stay updated



# EDUCATION

Register for expert-led training

- Instructor-led FreeNAS training
- Learn the ins and outs of FreeNAS
- Topic specific lessons

