Aaron Toponce

{ 2012 12 05 }

ZFS Administration, Part II- RAIDZ

Table of Contents

The previous post introduced readers to the concept of VDEVs with ZFS. This post continues the topic discusing the RAIDZ VDEVs in great detail.

Standards Parity RAID

To understand RAIDZ, you first need to understand parity-based RAID levels, such as RAID-5 and RAID-6. Let's discuss the standard RAID-5 layout. You need a minimum of 3 disks for a proper RAID-5 array. On two disks, the data is striped. A parity bit is then calculated such than the XOR of all three stripes in the set is calculated to zero. The parity is then written to disk. This allows you to suffer one disk failure, and recalculate the data. Further, in RAID-5, no single disk in the array is dedicated for the parity data. Instead, the parity is distributed throughout all of the disks. Thus, any disk can fail, and the data can still be restored.

However, we have a problem. Suppose that you write the data out in the RAID-5 stripe, but a power outtage occurs before you can write the parity. You now have inconsistent data. Jeff Bonwick, the creator of ZFS, refers to this as a "RAID-5 write hole". In reality, it's a problem, no matter how small, for all parity-based RAID arrays. If there exists any possibility that you can write the data blocks without writing the parity bit, then we have the "write hole". What sucks, is the software-based RAID is not aware that a problem exists. Now, there are software work-arounds to identify that the parity is inconsistent with the data, but they're slow, and not reliable. As a result, software-based RAID has fallen out of favor with storage administrators. Rather, expensive (and failure-prone) hardware cards, with battery backups on the card, have become commonplace.

There is also a big performance problem to deal with. If the data being written to the stripe is smaller than the stripe size, then the data must be read on the rest of the stripe, and the parity recalculated. This causes you to read and write data that is not pertinent to the application. Rather than reading only live, running data, you spend a great deal of time reading "dead" or old data. So, as a result, expensive battery-backed NVRAM hardware RAID cards can hide this latency from the user, while the NVRAM buffer fills working on this stripe, until it's been flushed to disk.

In both cases, the RAID-5 write hole, and writing data to disk that is smaller than the stripe size, the atomic transactional nature of ZFS does not like the hardware solutions, as it's impossible, and does not like existing software solutions as it opens up the possibility of corrupted data. So, we need to rethink parity-based RAID.

ZFS RAIDZ

Enter RAIDZ. Rather than the stripe width be statically set at creation, the stripe width is dynamic. Every block transactionally flushed to disk is its own stripe width. Every RAIDZ write is a full stripe write. Further, the parity bit is flushed with the stripe simultaneously, completely eliminating the RAID-5 write hole. So, in the event of a power failure, you either have the latest flush of data, or you don't. But, your disks will not be inconsistent.

RAID-5			
A1	A2	A3	Ap
B1	B2	Bp	B3
C1	Cp	C2	C3
Dp	D1	D2	D3
E1	E2	E3	Ep
F1	F2	Fp	F3
G1	Gp	G2	G3
HP	H1	H2	H3

RAID-Z1			
A1	A2	A3	Ap
A4	A5	A6	Ap'
B1	B2	Bp	C1
C2	C3	Cp	D1
Dp	E1	E2	E3
Ep	E4	Ep'	F1
Fp	G1	G2	G3
Gp	H1	H2	Hp

Demonstrating the dynamic stripe size of RAIDZ

There's a catch however. With standardized parity-based RAID, the logic is as simple as "every disk XORs to zero". With dynamic variable stripe width, such as RAIDZ, this doesn't work. Instead, we must pull up the ZFS metadata to determine RAIDZ geometry on

every read. If you're paying attention, you'll notice the impossibility of such if the filesystem and the RAID are separate products; your RAID card knows nothing of your filesystem, and vice-versa. This is what makes ZFS win.

Further, because ZFS knows about the underlying RAID, performance isn't an issue unless the disks are full. Reading filesystem metadata to construct the RAID stripe means only reading live, running data. There is no worry about reading "dead" data, or unallocated space. So, metadata traversal of the filesystem can actually be faster in many respects. You don't need expensive NVRAM to buffer your write, nor do you need it for battery backup in the event of RAID write hole. So, ZFS comes back to the old promise of a "Redundant Array of Inexpensive Disks". In fact, it's highly recommended that you use cheap SATA disk, rather than expensive fiber channel or SAS disks for ZFS.

Self-healing RAID

This brings us to the single-largest reason why I've become such a ZFS fan. ZFS can detect silent errors, and fix them on the fly. Suppose for a moment that there is bad data on a disk in the array, for whatever reason. When the application requests the data, ZFS constructs the stripe as we just learned, and compares each block against a default checksum in the metadata, which is currently fletcher4. If the read stripe does not match the checksum, ZFS finds the corrupted block, it then reads the parity, and fixes it through combinatorial reconstruction. It then returns good data to the application. This is all accomplished in ZFS itself, without the help of special hardware. Another aspect of the RAIDZ levels is the fact that if the stripe is longer than the disks in the array, if there is a disk failure, not enough data with the parity can reconstruct the data. Thus, ZFS will mirror some of the data in the stripe to prevent this from happening.

Again, if your RAID and filesystem are separate products, they are not aware of each other, so detecting and fixing silent data errors is not possible. So, with that out of the way, let's build some RAIDZ pools. As with my previous post, I'll be using 5 USB thumb drives /dev/sde, /dev/sdf, /dev/sdg, /dev/sdh and /dev/sdi which are all 8 GB in size.

RAIDZ-1

RAIDZ-1 is similar to RAID-5 in that there is a single parity bit distributed across all the disks in the array. The stripe width is variable, and could cover the exact width of disks in the array, fewer disks, or more disks, as evident in the image above. This still allows for one disk failure to maintain data. Two disk failures would result in data loss. A minimum of 3 disks should be used in a RAIDZ-1. The capacity of your storage will be the number of disks in your array times the storage of the smallest disk, minus one disk for parity storage (there is a caveat to zpool storage sizes I'll get to in another post). So in my example, I should have roughly 16 GB of usable disk.

To setup a zpool with RAIDZ-1, we use the "raidz1" VDEV, in this case using only 3 USB drives:

```
# zpool create tank raidz1 sde sdf sdg
# zpool status tank
  pool: pool
 state: ONLINE
 scan: none requested
config:

        NAME          STATE     READ WRITE CKSUM
        pool          ONLINE       0     0     0
          raidz1-0    ONLINE       0     0     0
            sde       ONLINE       0     0     0
            sdf       ONLINE       0     0     0
            sdg       ONLINE       0     0     0

errors: No known data errors
```

Cleanup before moving on, if following in your terminal:

```
# zpool destroy tank
```

RAIDZ-2

RAIDZ-2 is similar to RAID-6 in that there is a dual parity bit distributed across all the disks in the array. The stripe width is variable, and could cover the exact width of disks in the array, fewer disks, or more disks, as evident in the image above. This still allows for two disk failures to maintain data. Three disk failures would result in data loss. A minimum of 4 disks should be used in a RAIDZ-2. The capacity of your storage will be the number of disks in your array times the storage of the smallest disk, minus two disks for parity storage. So in my example, I should have roughly 16 GB of usable disk.

To setup a zpool with RAIDZ-2, we use the "raidz2" VDEV:

```
# zpool create tank raidz2 sde sdf sdg sdh
# zpool status tank
  pool: pool
 state: ONLINE
 scan: none requested
config:

        NAME          STATE     READ WRITE CKSUM
```

```
        pool          ONLINE       0     0     0
          raidz2-0    ONLINE       0     0     0
            sde       ONLINE       0     0     0
            sdf       ONLINE       0     0     0
            sdg       ONLINE       0     0     0
            sdh       ONLINE       0     0     0

errors: No known data errors
```

Cleanup before moving on, if following in your terminal:

```
# zpool destroy tank
```

RAIDZ-3

RAIDZ-3 does not have a standardized RAID level to compare it to. However, it is the logical continuation of RAIDZ-1 and RAIDZ-2 in that there is a triple parity bit distributed across all the disks in the array. The stripe width is variable, and could cover the exact width of disks in the array, fewer disks, or more disks, as evident in the image above. This still allows for three disk failures to maintain data. Four disk failures would result in data loss. A minimum of 5 disks should be used in a RAIDZ-3. The capacity of your storage will be the number of disks in your array times the storage of the smallest disk, minus three disks for parity storage. So in my example, I should have roughly 16 GB of usable disk.

To setup a zpool with RAIDZ-3, we use the "raidz3" VDEV:

```
# zpool create tank raidz3 sde sdf sdg sdh sdi
# zpool status tank
  pool: pool
 state: ONLINE
 scan: none requested
config:

        NAME          STATE     READ WRITE CKSUM
        pool          ONLINE       0     0     0
          raidz3-0    ONLINE       0     0     0
            sde       ONLINE       0     0     0
            sdf       ONLINE       0     0     0
            sdg       ONLINE       0     0     0
            sdh       ONLINE       0     0     0
```

```
            sdi       ONLINE       0     0     0

errors: No known data errors
```

Cleanup before moving on, if following in your terminal:

```
# zpool destroy tank
```

Hybrid RAIDZ

Unfortunatly, parity-based RAID can be slow, especially when you have many disks in a single stripe (say a 48-disk JBOD). To speed things up a bit, it might not be a bad idea to chop up the single large RAIDZ VDEV into a stripe of multiple RAIDZ VDEVs. This will cost you usable disk space for storage, but can greatly increase performance. Of course, as with the previous RAIDZ VDEVs, the stripe width is variable within each nested RAIDZ VDEV. For each RAIDZ level, you can lose up to that many disks in each VDEV. So, if you have a stripe of three RAIDZ-1 VDEVs, then you can suffer a total of three disk failures, one disk per VDEV. Usable space would be calculated similarly. In this example, you would lose three disks due to parity storage in each VDEV.

To illustrate this concept, let's suppose we have a 12-disk storage server, and we want to lose as little disk as possible while maximizing performance of the stripe. So, we'll create 4 RAIDZ-1 VDEVs of 3 disks each. This will cost us 4 disks of usable storage, but it will also give us the ability to suffer 4 disk failures, and the stripe across the 4 VDEVs will increase performance.

To setup a zpool with 4 RAIDZ-1 VDEVs, we use the "raidz1" VDEV 4 times in our command. Notice that I've added emphasis on when to type "raidz1" in the command for clarity:

```
# zpool create tank raidz1 sde sdf sdg raidz1 sdh sdi sdj raidz1 sdk sdl sdm raidz1 sdn sdo sdp
# zpool status tank
  pool: pool
 state: ONLINE
 scan: none requested
config:

        NAME          STATE     READ WRITE CKSUM
        pool          ONLINE       0     0     0
          raidz1-0    ONLINE       0     0     0
            sde       ONLINE       0     0     0
            sdf       ONLINE       0     0     0
            sdg       ONLINE       0     0     0
          raidz1-1    ONLINE       0     0     0
```

```
            sdh       ONLINE       0     0     0
            sdi       ONLINE       0     0     0
            sdj       ONLINE       0     0     0
          raidz1-2    ONLINE       0     0     0
            sdk       ONLINE       0     0     0
            sdl       ONLINE       0     0     0
            sdm       ONLINE       0     0     0
          raidz1-3    ONLINE       0     0     0
            sdn       ONLINE       0     0     0
            sdo       ONLINE       0     0     0
            sdp       ONLINE       0     0     0

errors: No known data errors
```

Notice now that there are four RAIDZ-1 VDEVs. As mentioned in a previous post, ZFS stripes across VDEVs. So, this setup is essentially a RAIDZ-1+0. Each RAIDZ-1 VDEV will receive 1/4 of the data sent to the pool, then each striped piece will be further striped across the disks in each VDEV. Nested VDEVs can be a great way to keep performance alive and well, long after the pool has been massively fragmented.

Cleanup before moving on, if following in your terminal:

```
# zpool destroy tank
```

Some final thoughts on RAIDZ

Various recommendations exist on when to use RAIDZ-1/2/3 and when not to. Some people say that a RAIDZ-1 and RAIDZ-3 should use an odd number of disks. RAIDZ-1 should start with 3 and not exceed 7 disks in the array, while RAIDZ-3 should start at 7 and not exceed 15. RAIDZ-2 should use an even number of disks, starting with 6 disks and not exceeding 12. This is to ensure that you have an even number of disks the data is actually being written to, and to maximize performance on the array.

~~Instead, in my opinion, you should keep your RAIDZ array at a low power of 2 plus parity. For RAIDZ-1, this is 3, 5 and 9 disks. For RAIDZ-2, this is 4, 6, 10, and 18 disks. For RAIDZ-3, this is 5, 7, 11, and 19 disks.~~ If going north of these recommendations, I would use RAID-1+0 setups personally. This is largely due to the time it will take to rebuild the data (called "resilvering"- a post coming in a bit). Because calculating the parity bit is so expensive, the more disks in the RAIDZ arary, the more expensive this operation will be, as compared to RAID-1+0.

UPDATE: I no longer recommend the "power of 2 plus parity" setup. It's mostly a myth. See http://blog.delphix.com/matt/2014/06/06/zfs-stripe-width/ for a good argument as to why.

Further, I've seen recommendations on the sizes that the disks should be, saying not to exceed 1 TB per disk for RAIDZ-1, 2 TB per disk for RAIDZ-2 and 3 TB per disk for RAIDZ-3. For sizes exceeding these values, you should use 2-way or 3-way mirrors with striping. Whether or not there is any validity to these claims, I cannot say. But, I can tell you that with the fewer number of disks, you should use a RAID level that accomadates your shortcomings. In a 4-disk RAID array, as I have above, calculating multiple parity bits can kill performance. Further, I could suffer at most two disk failures (if using RAID-1+0 or RAIDZ-2). RAIDZ-1 meets somewhere in the middle, where I can suffer a disk failure while stil maintaining a decent level of performance. If I had say 12 disks in the array, then maybe a RAIDZ-1+0 or RAIDZ-3 would be better suited, as the chances of suffering multiple disk failures increases.

Ultimately, you need to understand your storage problem and benchmark your disks. Put them in various RAID configurations, and use a utility such as IOZone 3 to benchmark and stress the array. You know what data you are going to store on the disk. You know what sort of hardware the disks are being installed into. You know what sort of performance you are looking for. It's your decision, and if you spend your time doing research, homework and sleuthing, you will arrive at the right decision. There may be "best practices", but they only go as far as your specific situation.

Lastly, in terms of performance, mirrors will always outperform RAIDZ levels. On both reads and writes. Further, RAIDZ-1 will outperform RAIDZ-2, which it turn will outperform RAIDZ-3. The more parity bits you have to calculate, the longer it's going to take to both read and write the data. Of course, you can always add striping to your VDEVs to maximize on some of this performance. Nested RAID levels, such as RAID-1+0 are considered "the Cadillac of RAID levels" due to the flexibility in which you can lose disks without parity, and the throughput you get from the stripe. So, in a nutshell, from fastest to slowest, your non-nested RAID levels will perform as:

- RAID-0 (fastest)
- RAID-1
- RAIDZ-1
- RAIDZ-2
- RAIDZ-3 (slowest)

Posted by Aaron Toponce on Wednesday, December 5, 2012, at 6:00 am. Filed under Debian, Linux, Ubuntu, ZFS. Follow any responses to this post with its

comments RSS feed. You can post a comment or trackback from your blog. For

IM, Email or Microblogs, here is the Shortlink.

{ 31 } Comments

1. Jon | December 5, 2012 at 9:11 am | Permalink

   Thanks for the pair of articles. I've started messed around with ZFS on one of the scrap server that sits next to my desk. I've read the docs and FAQs, but it's good to see a different perspective of the basic setup.

   I look forward to your next article since, as of last night, one of the drives in the test server has started racking up SMART errors at an alarming rate. I guess I'll get to test resilvering in the real case and not just by faking a drive failure. :O

2. Aaron Toponce | December 5, 2012 at 9:40 am | Permalink

   Np. However, the 3rd post will be covering more VDEVs (there is an order to my chaos). In this case, I'll be covering the L2ARC and the ZIL. Hope to have it up tomorrow morning. Might be a day late though.

3. David | December 5, 2012 at 2:58 pm | Permalink

   Very helpful articles! I've been using ZFS for the past year, and have been extremely impressed by it. Looking forward to your L2ARC and ZIL article, as that's something we'll definitely be wanting to add in the near future.

4. Mark | December 7, 2012 at 11:00 pm | Permalink

   Aaron, I've enjoyed reading the article. Is it really a bad idea to use 5 disks in a RAID-Z2 arrangement? I have 5 x 2TB disks that I want to use in my FreeNAS box, and prefer to have dual parity (rather than RAID-Z1).

5. Aaron Toponce | December 8, 2012 at 7:43 am | Permalink

   "A bad idea", no. However, it's also not optimized. My hypervisors are using RAIDZ-1 with 4 disks, as I needed the space. My motherboard does not have enough SATA ports for 5 disks, and I need more space than what 3 disks would give. Thus, RAIDZ-1 on four disks it is. You do what you can.

6. boneidol | December 28, 2012 at 7:36 pm | Permalink

"In relatiy" <- trivial typo

7. Aaron Toponce | December 29, 2012 at 6:24 am | Permalink

   Fixed. Thanks!

8. boneidol | December 28, 2012 at 7:42 pm | Permalink

   "Instead, in my opinion, you should keep your RAIDZ array at a low power of 2 plus parity. For RAIDZ-1, this is 3, 5 and 9 disks. For RAIDZ-2, this is 4, 8 and 16 disks. For RAIDZ-3, this is 5, 9 and 17 disks"

   hi I don't understand these numbers above

   Z1 = 2^1 + 1 , 2 ^2 + 1 , 2^3 +1 = 3,5,9
   Z2 = 2^1 + 2 , 2^2 +2 , 2^3 +2 = 4,6,10
   Z3 = 2^1 + 3 , 2^2 +3 , 2^3 +3 = 5,7,11

   Sorry!

9. Alvin | February 2, 2013 at 9:53 pm | Permalink

   Okay here's one for you, I can't find ANY documentation ANYWHERE for using brackets (parentheses) to describe what drives to select when creating a zpool. For example, I am in a VERY sticky situation with money and physical drive constraints. I have figured out a method to make the best use of what I have but it results in a pretty unorthodox (yet completely redundant and failproof [1 drive] way) of getting it all to work AND maximize the use of my motherboard's ports to make it completly expandable in the future. I am basically creating a single-vdev pool containing a bunch of different raid levels, mirrors, and stripes.

   HOWEVER, this is how I have to do it, because of hardware constraints.
   If you were to imagine how to use the zpool create, this is how it would look USING BRACKETS. BUT THERE IS NO MENTION OF HOW TO USE BRACKETS PROPERLY in any zfs documentation. Basically either brackets, commas, &&s, etc, anything that would give me the desired affect.

   zpool create mycoolpool RAIDZ1 ((mirror A B) (mirror C D) (mirror E F) (G) (stripe H, I) (stripe J, K, L) (M))

   Yes I have 7 1TB 'blocks' or 'chunks' in a RAIDZ1, each consisting of different configurations.

You see, if I were to do this without the brackets, it would create this mess:
zpool create mycoolpool RAIDZ1 mirror a b mirror c d mirror e f g h i j k l m
^^Basically you see here that I would end up with a RAIDZ1 across 3 mirrors, the third of which consisting of a redundancy level such that 8 drives could fail... not what I want.

And yes, I have indeed seen all the warnings and read countless people say "you shouldn't" but NEVER have I seen anyone deny that it could be done and NEVER have I seen anyone actually answer on HOW to do it.

I've made up my mind that this is the method and approach that I need to take so please heed your warnings as much as you can as they will be said in vain.

Thank you very much in advance for a response!!!

10. Aaron Toponce | February 7, 2013 at 10:24 am | Permalink

    No, this is not possible. Other than disks and files, you cannot nest VDEVs. ZFS stripes across RAIDZ and mirror VDEVs, and there's no way around it. You need to rethink your storage.

11. ssl | March 26, 2013 at 11:54 am | Permalink

    I don't quite understand how zfs could recover from certain single disk failures in your example (picture) .. say for example you lost the last drive in your raidz-1 configuration as shown. for the long stripe (A) you lose the parity bit as well as the data in block A4... How could this possibly be recovered, unless zfs puts additional parity blocks in for all stripes whose length exceeds the number of disks??

12. Aaron Toponce | March 27, 2013 at 1:44 pm | Permalink

    Correct. The image isn't 100% accurate. I may fix it, but yes. If you lose too much of a single stripe, then you can't recreate the data. For each stripe written, and this is where my image needs to be updated, a parity bit is written. So, if a stripe crosses the disks twice, then there will be extra parity bits.

    Thanks for pointing this out.

13. Veniamin | April 30, 2013 at 12:54 am | Permalink

    Thanks for articke.
    I wonder how RAIDZ will work with two or more parity stripes.

I think that in the case of data is longer than recsize x n_data_disks, raidz slpits it into several writes.

14. Aaron Toponce | December 20, 2013 at 11:02 pm | Permalink

    I've updated the image (finally) to reflect the inconsistencies I had before.

15. Heny | January 14, 2014 at 9:34 am | Permalink

    ZFS RAIDZ as declustered RAID, how to acheive it?

16. Chris | April 3, 2014 at 2:01 pm | Permalink

    Great articles! Thanks a lot. I was wondering if you have any source for the comments on maximum drive size for the various raidz types? I am very interested why someone thinks maximum 2TB for raidz-2 (as I want to create an array of 8 disks, each 4TB large in a raidz-2 configuration).

17. Aaron Toponce | April 12, 2014 at 3:36 pm | Permalink

    I haven't seen anything regarding maximum drive size. Of course, you need to benchmark your own system, but the more storage you have, the more storage you have. Generally speaking too, the more spindles you have, the better performance will be.

18. TK | July 5, 2014 at 12:21 am | Permalink

    Typo:
    # zpool create tank raidze sde sdf sdg sdh sdi
    should read:
    # zpool create tank raidz3 sde sdf sdg sdh sdi

    That aside, these are all very informative ZFS articles, as are most of your others on th evariety of topics you cover.
    Regards,
    --TK

19. Aaron Toponce | July 5, 2014 at 9:12 am | Permalink

    Fixed! Thanks for the edit.

20. John Naggets | January 30, 2015 at 10:43 am | Permalink

I am still hesitating about the size of my RAIDZ-2 array. Would it be ok to use 12 disks on a RAIDZ-2 array? isn't that too much? and what about using 9 disks in a RAIDZ-2 array? does the rule of an even number for RAIDZ-2 still apply nowadays?

21. Aaron Toponce | February 17, 2015 at 1:46 pm | Permalink

    > I am still hesitating about the size of my RAIDZ-2 array. Would it be ok to use 12 disks on a RAIDZ-2 array? isn't that too much? and what about using 9 disks in a RAIDZ-2 array? does the rule of an even number for RAIDZ-2 still apply nowadays?

    I wouldn't do RAIDZ2 personally. With 12 disks, I would do RAIDZ1 of 3 disks each. Thus, I would have 4 RAIDZ1 VDEVs:

```
# zpool status pthree
  pool: pthree
 state: ONLINE
  scan: none requested
config:

        NAME             STATE     READ WRITE CKSUM
        pthree           ONLINE       0     0     0
          raidz1-0       ONLINE       0     0     0
            /tmp/file1   ONLINE       0     0     0
            /tmp/file2   ONLINE       0     0     0
            /tmp/file3   ONLINE       0     0     0
          raidz1-1       ONLINE       0     0     0
            /tmp/file4   ONLINE       0     0     0
            /tmp/file5   ONLINE       0     0     0
            /tmp/file6   ONLINE       0     0     0
          raidz1-2       ONLINE       0     0     0
            /tmp/file7   ONLINE       0     0     0
            /tmp/file8   ONLINE       0     0     0
            /tmp/file9   ONLINE       0     0     0
          raidz1-3       ONLINE       0     0     0
            /tmp/file10  ONLINE       0     0     0
            /tmp/file11  ONLINE       0     0     0
            /tmp/file12  ONLINE       0     0     0

errors: No known data errors
```

At least then, you can keep your performance up, while maintaining one disk failure in each VDEV (a total of 3 disk failures maximum). It only comes at the cost of losing 1/3 of the raw disk space, which IMO, isn't that bad.

22. John Naggets | February 18, 2015 at 11:26 am | Permalink

    Thanks for your extensive answer! Actually I was planning to do a RAIDZ-2 of 12 disks because my server can host up to 36 disks. So my plan would be to start with one RAIDZ-2 vdev of 12 disks and then increase the storage always by 12 disks, ending up with 3 RAIDZ-2 vdevs of 12 disks each.

    Or would you still recommend having 12 RAIDZ-1 vdevs of 3 disks each like you mention in your answer? The thing is that with your setup I would be "loosing" in total 12 disks (parity) whereas with my config I would be only "loosing" 6 disks.

23. Aaron Toponce | February 18, 2015 at 1:03 pm | Permalink

    > would you still recommend having 12 RAIDZ-1 vdevs of 3 disks each like you mention in your answer?

    That depends on what you plan on doing with your pool, how much space you need, and how you expect it to perform. If it's just a backup server, that runs nightly backup jobs, and is guaranteed to finish before the next cycle, then performance probably isn't *that* big of a deal (until you need to do a restore at least).

    Regardless, I can't fully answer that. I would build the pool multiple ways, banchmark it, stress test it, fill it and refill it, fail drives, and over all, put it through a stringent series of tests, and see which configuration would be the best for you. Parity-based RAID can be a performance killer but it can be worth it in some scenarios.

24. Ben | May 29, 2015 at 6:14 pm | Permalink

    Thank you so much for putting all this information up. I had spent a good week reading up on ZFS and how to use it, but was still confused beyond belief. I am a long term windows user and I'm just getting into linux and what it can do. Your postings here are laid out so well that it helped me understand how everything works. Thank you again!

25. Rares | June 30, 2015 at 11:52 pm | Permalink

    Amassing documentation. If I'l ever meet you, the beer is on me:D

26. Jim | July 22, 2015 at 11:07 am | Permalink

Thanks for the ZFS docs. What's the best practice for creating a zpool for use in a RAID1+0 or RAIDZn array from the point of view of future drive replacement?

What is the likelihood of a replacement drive having a slightly smaller actual capacity than the drive it's replacing? Since we cannot shrink a zpool once created, what would happen if a replacement drive is found to be 1 sector smaller than the failed drive? I assume ZFS issues an error saying that it cannot populate the new drive?

Is it best practice to manually partition drives prior to adding to the initial zpool so that all members of the pool are a known, precise size? Or is this generally a non-issue?

27. Frank | March 1, 2016 at 7:25 pm | Permalink

Hi Aaron,
Thank you once again for your great summaries.
I'm embarking on building a large array for scientific data storage (actually I'm building two identical arrays, one for backup).
I wonder if the plan is sane:

The arrays will require around 100TB of storage eventually but I'm starting with 8x HGST 8TB SAS disks.

So I was thinking of doing a striped set of two RAIDZ1 vdevs.
If my calculations are right, this gives 64TB-(2x8TB)=48TB Storage
The data will be backed up to a clone server using zfs send nightly
and also to tape.

NAME
bigpool
raidz1-0
8TB disk1
8TB disk2
8TB disk3
8TB disk4
raidz1-0
8TB disk1
8TB disk2
8TB disk3
8TB disk4

logs
mirror-1
1.2TB SSD
1.2TB SSD

In due course, I'd add another 8 disks (again as 2x striped RAIDZ1 vdevs)
for a total storage capacity of 96TB (close enough to 100TB).

I'm a little worried that recovering from a disk failure on a vdev with 4x 8TB disks may be a bit risky. The other two options I considered were:
- 8 disk RAIDZ3 array (eventually striped with another 2 of these)
- striped mirrors (but the capacity loss is expensive)

I'd be curious to hear your recommendations

28. Eric | September 10, 2016 at 4:17 am | Permalink

    How come it seems like most documentation say mirrored is always faster than raidz(n), but benchmarks always seem to show the opposite? (https://pthree.org/2012/12/05/zfs-administration-part-ii-raidz/) (https://calomel.org/zfs_raid_speed_capacity.html)

29. gsalerni | January 13, 2017 at 11:08 am | Permalink

    re. Alvins post (9) about trying to assemble a raidZ pool made up of 1tb vdevs which were in turn a variety of single disks, mirrors and stripes. Although you can't nest vdevs (other than disks and files) - could he not use madam to construct the various 1tb metadisks using md mirrors and stripes as required and then create a zfs raidz out of those? I imagine that wouldn't perform great but would it work? zfs wouldn't care that the raw disks were in fact meta disks would it?

30. TMS | March 5, 2017 at 10:47 am | Permalink

    Very nice article, but you are incorrect wehn you say mirror is ALWAYS faster. No it isn't. For sequential reads Raidz is faster. Same with writes. IOPS are always faster on a mirror.

31. xaoc | August 22, 2017 at 3:04 am | Permalink

    I have strange situation and can't explain it . I will appreciate your comment on bellow setup:
    zpool list
    NAME SIZE ALLOC FREE EXPANDSZ FRAG CAP DEDUP HEALTH ALTROOT

```
test_3x3s 327T 1.11M 327T - 0% 0% 1.00x ONLINE -
dmadm@s1349014530:~$ sudo zpool status
pool: test_3x3s
state: ONLINE
scan: none requested
config:

NAME STATE READ WRITE CKSUM
test_3x3s ONLINE 0 0 0
raidz3-0 ONLINE 0 0 0
sdc ONLINE 0 0 0
sdd ONLINE 0 0 0
sde ONLINE 0 0 0
sdf ONLINE 0 0 0
sdg ONLINE 0 0 0
sdh ONLINE 0 0 0
sdi ONLINE 0 0 0
sdj ONLINE 0 0 0
sdk ONLINE 0 0 0
sdl ONLINE 0 0 0
sdm ONLINE 0 0 0
sdn ONLINE 0 0 0
raidz3-1 ONLINE 0 0 0
sdo ONLINE 0 0 0
sdp ONLINE 0 0 0
sdq ONLINE 0 0 0
sdr ONLINE 0 0 0
sds ONLINE 0 0 0
sdt ONLINE 0 0 0
sdu ONLINE 0 0 0
sdv ONLINE 0 0 0
sdw ONLINE 0 0 0
sdx ONLINE 0 0 0
sdy ONLINE 0 0 0
sdz ONLINE 0 0 0
raidz3-2 ONLINE 0 0 0
```

```
sdaa ONLINE 0 0 0
sdab ONLINE 0 0 0
sdac ONLINE 0 0 0
sdad ONLINE 0 0 0
sdae ONLINE 0 0 0
sdaf ONLINE 0 0 0
sdag ONLINE 0 0 0
sdah ONLINE 0 0 0
sdai ONLINE 0 0 0
sdaj ONLINE 0 0 0
sdak ONLINE 0 0 0
sdal ONLINE 0 0 0

errors: No known data errors
df -h
Filesystem Size Used Avail Use% Mounted on
udev 189G 0 189G 0% /dev
tmpfs 38G 850M 37G 3% /run
/dev/md0 103G 1.9G 96G 2% /
tmpfs 189G 0 189G 0% /dev/shm
tmpfs 5.0M 0 5.0M 0% /run/lock
tmpfs 189G 0 189G 0% /sys/fs/cgroup
tmpfs 38G 0 38G 0% /run/user/1002
test_3x3s 231T 256K 231T 1% /test_3x3s
################################################################################################################
zpool list
NAME SIZE ALLOC FREE EXPANDSZ FRAG CAP DEDUP HEALTH ALTROOT
test_3x3s 326T 1.11M 326T - 0% 0% 1.00x ONLINE -
dmadm@s1349014530:~$ df -h
Filesystem Size Used Avail Use% Mounted on
udev 189G 0 189G 0% /dev
tmpfs 38G 858M 37G 3% /run
/dev/md0 103G 1.9G 96G 2% /
tmpfs 189G 0 189G 0% /dev/shm
tmpfs 5.0M 0 5.0M 0% /run/lock
tmpfs 189G 0 189G 0% /sys/fs/cgroup
```

```
tmpfs 38G 0 38G 0% /run/user/1002
test_3x3s 230T 256K 230T 1% /test_3x3s
zpool status
pool: test_3x3s
state: ONLINE
scan: none requested
config:

NAME STATE READ WRITE CKSUM
test_3x3s ONLINE 0 0 0
raidz3-0 ONLINE 0 0 0
sdc ONLINE 0 0 0
sdd ONLINE 0 0 0
sde ONLINE 0 0 0
sdf ONLINE 0 0 0
sdg ONLINE 0 0 0
sdh ONLINE 0 0 0
sdi ONLINE 0 0 0
sdj ONLINE 0 0 0
sdk ONLINE 0 0 0
sdl ONLINE 0 0 0
sdm ONLINE 0 0 0
sdn ONLINE 0 0 0
sdo ONLINE 0 0 0
sdp ONLINE 0 0 0
sdq ONLINE 0 0 0
sdr ONLINE 0 0 0
sds ONLINE 0 0 0
sdt ONLINE 0 0 0
raidz3-1 ONLINE 0 0 0
sdu ONLINE 0 0 0
sdv ONLINE 0 0 0
sdw ONLINE 0 0 0
sdx ONLINE 0 0 0
sdy ONLINE 0 0 0
sdz ONLINE 0 0 0
```

sdaa ONLINE 0 0 0
sdab ONLINE 0 0 0
sdac ONLINE 0 0 0
sdad ONLINE 0 0 0
sdae ONLINE 0 0 0
sdaf ONLINE 0 0 0
sdag ONLINE 0 0 0
sdah ONLINE 0 0 0
sdai ONLINE 0 0 0
sdaj ONLINE 0 0 0
sdak ONLINE 0 0 0
sdal ONLINE 0 0 0

In few words ... If I undesrtand it correctly:
2 VDEVs RAIDZ3 should use 6 disks for parity (3 for each VDEV)
3 VDEVs RAIDZ3 should use 9 disks for parity (3 for each VDEV)
And it is logical to have less usable space with 3 VDEVs compared with 2 VDEVs, but practicaly it seems that with 2 VDEVs configuration I have less usable space?

{ 13 } Trackbacks

1. Aaron Toponce : ZFS Administration, Part III- The ZFS Intent Log | December 6, 2012 at 6:00 am | Permalink

   [...] The previous post about using ZFS with GNU/Linux concerned covering the three RAIDZ virtual devices .... This post will cover another VDEV- the ZFS Intent Log, or the ZIL. [...]

2. Aaron Toponce : ZFS Administration, Part I- VDEVs | December 13, 2012 at 5:59 am | Permalink

   [...] RAIDZ [...]

3. Aaron Toponce : Install ZFS on Debian GNU/Linux | December 13, 2012 at 6:05 am | Permalink

   [...] RAIDZ [...]

4. Aaron Toponce : ZFS Administration, Part VI- Scrub and Resilver | December 13, 2012 at 6:07 am | Permalink

[...] RAIDZ [...]

5. Aaron Toponce : ZFS Administration, Part XII- Snapshots and Clones | December 20, 2012 at 8:06 am | Permalink

   [...] RAIDZ [...]

6. Aaron Toponce : ZFS Administration, Part VIII- Zpool Best Practices and Caveats | December 20, 2012 at 8:07 am | Permalink

   [...] RAIDZ [...]

7. Aaron Toponce : ZFS Administration, Part XI- Compression and Deduplication | January 7, 2013 at 9:23 pm | Permalink

   [...] RAIDZ [...]

8. Aaron Toponce : ZFS Administration, Part V- Exporting and Importing zpools | January 7, 2013 at 9:25 pm | Permalink

   [...] RAIDZ [...]

9. Aaron Toponce : ZFS Administration, Part VII- Zpool Properties | February 20, 2013 at 8:33 am | Permalink

   [...] RAIDZ [...]

10. Aaron Toponce : ZFS Administration, Part IX- Copy-on-write | April 19, 2013 at 4:57 am | Permalink

    [...] RAIDZ [...]

11. Aaron Toponce : ZFS Administration, Appendix A- Visualizing The ZFS Intent LOG (ZIL) | April 19, 2013 at 5:03 am | Permalink

    [...] RAIDZ [...]

12. Aaron Toponce : ZFS Administration, Part XIII- Sending and Receiving Filesystems | July 2, 2013 at 7:24 am | Permalink

    […] RAIDZ […]

13. Aaron Toponce : ZFS Administration, Appendix B- Using USB Drives | July 8, 2013 at 10:08 pm | Permalink

    […] RAIDZ […]