Aaron Toponce

{ 2012 12 11 }

ZFS Administration, Part VI- Scrub and Resilver

Table of Contents

Standard Validation

In GNU/Linux, we have a number of filesystem checking utilities for verifying data integrity on the disk. This is done through the "fsck" utility. However, it has a couple major drawbacks. First, you must fsck the disk offline if you are intending on fixing data errors. This means downtime. So, you must use the "umount" command to unmount your disks, before the fsck. For root partitions, this further means booting from another medium, like a CDROM or USB stick. Depending on the size of the disks, this downtime could take hours. Second, the filesystem, such as ext3 or ext4, knows nothing of the underlying data structures, such as LVM or RAID. You may only have a bad block on one disk, but a good block on another disk. Unfortunately, Linux software RAID has no idea which is good or bad, and from the perspective of ext3 or ext4, it will get good data if read from the disk containing the good block, and corrupted data from the disk containing the bad block, without any control over which disk to pull the data from, and fixing the

corruption. These errors are known as "silent data errors", and there is really nothing you can do about it with the standard GNU/Linux filesystem stack.

ZFS Scrubbing

With ZFS on Linux, detecting and correcting silent data errors is done through scrubbing the disks. This is similar in technique to ECC RAM, where if an error resides in the ECC DIMM, you can find another register that contains the good data, and use it to fix the bad register. This is an old technique that has been around for a while, so it's surprising that it's not available in the standard suite of journaled filesystems. Further, just like you can scrub ECC RAM on a live running system, without downtime, you should be able to scrub your disks without downtime as well. With ZFS, you can.

While ZFS is performing a scrub on your pool, it is checking every block in the storage pool against its known checksum. Every block from top-to-bottom is checksummed using an appropriate algorithm by default. Currently, this is the "fletcher4" algorithm, which is a 256-bit algorithm, and it's *fast*. This can be changed to using the SHA-256 algorithm, although it may not recommended, as calculating the SHA-256 checksum is more costly than fletcher4. However, because of SHA-256, you have a 1 in 2^256 or 1 in 10^77 chance that a corrupted block hashes to the same SHA-256 checksum. This is a 0.00000000000000000000000000000000000000000000000000000000000000000000000000000001% chance. For reference, uncorrected ECC memory errors will happen on about 50 orders of magnitude more frequently, with the most reliable hardware on the market. So, when scrubbing your data, the probability is that either the checksum will match, and you have a good data block, or it won't match, and you have a corrupted data block.

Scrubbing ZFS storage pools is not something that happens automatically. You need to do it manually, and it's highly recommended that you do it on a regularly scheduled interval. The recommended frequency at which you should scrub the data depends on the quality of the underlying disks. If you have SAS or FC disks, then once per month should be sufficient. If you have consumer grade SATA or SCSI, you should do once per week. You can schedule a scrub easily with the following command:

```
# zpool scrub tank
# zpool status tank
  pool: tank
 state: ONLINE
 scan: scrub in progress since Sat Dec  8 08:06:36 2012
    32.0M scanned out of 48.5M at 16.0M/s, 0h0m to go
    0 repaired, 65.99% done
config:

        NAME        STATE     READ WRITE CKSUM
```

```
        tank        ONLINE       0     0     0
          mirror-0  ONLINE       0     0     0
            sde     ONLINE       0     0     0
            sdf     ONLINE       0     0     0
          mirror-1  ONLINE       0     0     0
            sdg     ONLINE       0     0     0
            sdh     ONLINE       0     0     0
          mirror-2  ONLINE       0     0     0
            sdi     ONLINE       0     0     0
            sdj     ONLINE       0     0     0

errors: No known data errors
```

As you can see, you can get a status of the scrub while it is in progress. Doing a scrub can severely impact performance of the disks and the applications needing them. So, if for any reason you need to stop the scrub, you can pass the "-s" switch to the scrub subcommand. However, you should let the scrub continue to completion.

```
# zpool scrub -s tank
```

You should put something similar to the following in your root's crontab, which will execute a scrub every Sunday at 02:00 in the morning:

```
0 2 * * 0 /sbin/zpool scrub tank
```

Self Healing Data

If your storage pool is using some sort of redundancy, then ZFS will not only detect the silent data errors on a scrub, but it will also correct them if good data exists on a different disk. This is known as "self healing", and can be demonstrated in the following image. In my RAIDZ post, I discussed how the data is self-healed with RAIDZ, using the parity and a reconstruction algorithm. I'm going to simplify it a bit, and use just a two way mirror. Suppose that an application needs some data blocks, and in those blocks, on of them is corrupted. How does ZFS know the data is corrupted? By checking the SHA-256 checksum of the block, as already mentioned. If a checksum does not match on a block, it will look at my other disk in the mirror to see if a good block can be found. If so, the good block is passed to the application, then ZFS will fix the bad block in the mirror, so that it also passes the SHA-256 checksum. As a result, the application will always get good data, and your pool will always be in a good, clean, consistent state.

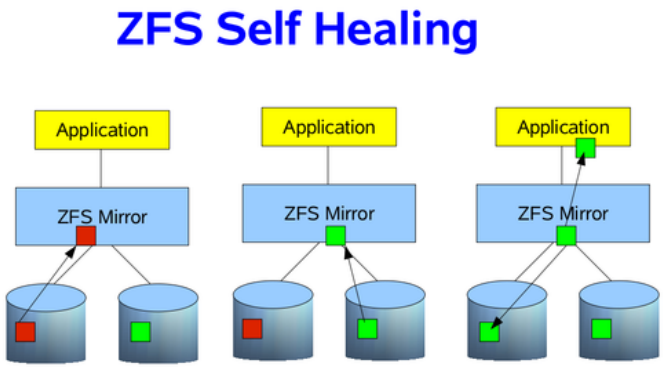

Image courtesy of root.cz, showing how ZFS self heals data.

Resilvering Data

Resilvering data is the same concept as rebuilding or resyncing data onto the new disk into the array. However, with Linux software RAID, hardware RAID controllers, and other RAID implementations, there is no distinction between which blocks are actually live, and which aren't. So, the rebuild starts at the beginning of the disk, and does not stop until it reaches the end of the disk. Because ZFS knows about the the RAID structure and the filesystem metadata, we can be smart about rebuilding the data. Rather than wasting our time on free disk, where live blocks are not stored, we can concern ourselves with ONLY those live blocks. This can provide significant time savings, if your storage pool is only partially filled. If the pool is only 10% filled, then that means only working on 10% of the drives. Win. Thus, with ZFS we need a new term than "rebuilding", "resyncing" or "reconstructing". In this case, we refer to the process of rebuilding data as "resilvering".

Unfortunately, disks will die, and need to be replaced. Provided you have redundancy in your storage pool, and can afford some failures, you can still send data to and receive data from applications, even though the pool will be in "DEGRADED" mode. If you have the luxury of hot swapping disks while the system is live, you can replace the disk without downtime (lucky you). If not, you will still need to identify the dead disk, and replace it. This can be a chore if you have many disks in your pool, say 24. However, most GNU/Linux operating system vendors, such as Debian or Ubuntu, provide a utility called "hdparm" that allows you to discover the serial number of all the disks in your pool. This is, of course, that the disk controllers are presenting that information to the Linux kernel, which they typically do. So, you could run something like:

```
# for i in a b c d e f g; do echo -n "/dev/sd$i: "; hdparm -I /dev/sd$i | awk '/Serial Number/ {print $3}'; done
/dev/sda: OCZ-9724MG8BII8G3255
/dev/sdb: OCZ-69ZO5475MT43KNTU
/dev/sdc: WD-WCAPD3307153
/dev/sdd: JP2940HD0K9RJC
/dev/sde: /dev/sde: No such file or directory
```

```
/dev/sdf: JP2940HD0SB8RC
/dev/sdg: S1D1C3WR
```

It appears that /dev/sde is my dead disk. I have the serial numbers for all the other disks in the system, but not this one. So, by process of elimination, I can go to the storage array, and find which serial number was not printed. This is my dead disk. In this case, I find serial number "JP2940HD01VLMC". I pull the disk, replace it with a new one, and see if /dev/sde is repopulated, and the others are still online. If so, I've found my disk, and can add it to the pool. This has actually happened to me twice already, on both of my personal hypervisors. It was a snap to replace, and I was online in under 10 minutes.

To replace an dead disk in the pool with a new one, you use the "replace" subcommand. Suppose the new disk also identifed itself as /dev/sde, then I would issue the following command:

```
# zpool replace tank sde sde
# zpool status tank
  pool: tank
 state: ONLINE
status: One or more devices is currently being resilvered.  The pool will
        continue to function, possibly in a degraded state.
action: Wait for the resilver to complete.
 scrub: resilver in progress for 0h2m, 16.43% done, 0h13m to go
config:

        NAME          STATE       READ WRITE CKSUM
        tank          DEGRADED       0     0     0
          mirror-0    DEGRADED       0     0     0
            replacing DEGRADED       0     0     0
            sde       ONLINE         0     0     0
            sdf       ONLINE         0     0     0
          mirror-1    ONLINE         0     0     0
            sdg       ONLINE         0     0     0
            sdh       ONLINE         0     0     0
          mirror-2    ONLINE         0     0     0
            sdi       ONLINE         0     0     0
            sdj       ONLINE         0     0     0
```

The resilver is analagous to a rebuild with Linux software RAID. It is rebuilding the data blocks on the new disk until the mirror, in this case, is in a completely healthy state. Viewing the status of the resilver will help you get an idea of when it will complete.

Identifying Pool Problems

Determining quickly if everything is functioning as it should be, without the full output of the "zpool status" command can be done by passing the "-x" switch. This is useful for scripts to parse without fancy logic, which could alert you in the event of a failure:

```
# zpool status -x
all pools are healthy
```

The rows in the "zpool status" command give you vital information about the pool, most of which are self-explanatory. They are defined as follows:

- pool- The name of the pool.
- state- The current health of the pool. This information refers only to the ability of the pool to provide the necessary replication level.
- status- A description of what is wrong with the pool. This field is omitted if no problems are found.
- action- A recommended action for repairing the errors. This field is an abbreviated form directing the user to one of the following sections. This field is omitted if no problems are found.
- see- A reference to a knowledge article containing detailed repair information. Online articles are updated more often than this guide can be updated, and should always be referenced for the most up-to-date repair procedures. This field is omitted if no problems are found.
- scrub- Identifies the current status of a scrub operation, which might include the date and time that the last scrub was completed, a scrub in progress, or if no scrubbing was requested.
- errors- Identifies known data errors or the absence of known data errors.
- config- Describes the configuration layout of the devices comprising the pool, as well as their state and any errors generated from the devices. The state can be one of the following: ONLINE, FAULTED, DEGRADED, UNAVAILABLE, or OFFLINE. If the state is anything but ONLINE, the fault tolerance of the pool has been compromised.

The columns in the status output, "READ", "WRITE" and "CHKSUM" are defined as follows:

- NAME- The name of each VDEV in the pool, presented in a nested order.
- STATE- The state of each VDEV in the pool. The state can be any of the states found in "config" above.
- READ- I/O errors occurred while issuing a read request.
- WRITE- I/O errors occurred while issuing a write request.
- CHKSUM- Checksum errors. The device returned corrupted data as the result of a read request.

Conclusion

Scrubbing your data on regular intervals will ensure that the blocks in the storage pool remain consistent. Even though the scrub can put strain on applications wishing to read or write data, it can save hours of headache in the future. Further, because you could have a "damaged device" at any time (see http://docs.oracle.com/cd/E19082-01/817-2271/gbbvf/index.html about damaged devices with ZFS), properly knowing how to fix the device, and what to expect when replacing one, is critical to storage administration. Of course, there is plenty more I could discuss about this topic, but this should at least introduce you to the concepts of scrubbing and resilvering data.

Posted by Aaron Toponce on Tuesday, December 11, 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.

{ 15 } Comments

1. Anca Emanuel | December 11, 2012 at 11:51 am | Permalink

   This is wrestling with the octopus ??
   Conclusion: the need for simple standard administation was ignored.

2. Aaron Toponce | December 11, 2012 at 12:45 pm | Permalink

   Huh?

3. eagle275 | February 18, 2013 at 8:36 am | Permalink

   I believe he meant :

   Put (self adhesive) Labels on each device , that contain Serial Number AND to which device the disk was "discovered" by linux - so you should have known "oh sde fails" - look at your array, find attached label "sde" and you have the missing disk, without trial and error